Universidade de Aveiro
2012

Departamento de Matemática

ALICE MANUELA VERGUEIRO PAIVA

CheckPGM: conversor para cablagens

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Matemática e Aplicações, ramo Ciências da Computação, realizada sob a orientação científica do Doutor Pedro Cruz, Professor Auxiliar do Departamento de Matemática da Universidade de Aveiro.

O júri

presidente

Prof.ª Doutora Maria Rosália Dinis Rodrigues
professora associada aposentada da Universidade de Aveiro

Prof. Doutor Luís Manuel Dias Coelho Soares Barbosa
professor associado da Universidade do Minho

Prof. Doutor João Pedro Antunes Ferreira da Cruz
professor auxiliar da Universidade de Aveiro

Engª. Eugénia Margarida Pinho Vinagre
profissional de engenharia III, Yazaki Saltano

Agradecimentos

Ao Professor Doutor Pedro Cruz, meu orientador, pela competência científica e acompanhamento ao longo projeto, pela paciência e disponibilidade, pelas correções, sugestões, que foram cruciais no decorrer do estágio, e pela orientação dada.

À Engenheira Eugénia Vinagre, minha orientadora na empresa, pela paciência e pelo empenho em ajudar-me a dominar conceitos novos, bem como pela disponibilidade e sugestões dadas ao longo de todo o projeto.

Aos meus colegas de trabalho, pelo apoio, pela ajuda e pelos bons momentos.

À minha família, pelo apoio incansável e palavras carinhosas que me acompanharam não só durante o estágio, mas durante toda a minha vida, que me ajudaram a superar as dificuldades que foram surgindo.

Por fim, ao João, meu namorado, pela generosidade e dedicação, pela força e segurança e acima de tudo, pela alegria nos dias de maior cansaço.

Palavras-Chave

Painéis elétricos, cablagens, SQL, ASP.NET, Expressões regulares, C#, CheckPGM.

Resumo

O presente documento foi desenvolvido no âmbito do estágio curricular do Mestrado em Matemática e Aplicações, ramo Ciências da Computação, da Universidade de Aveiro e tem como objetivo descrever pormenorizadamente o projeto realizado na empresa Yazaki Saltano, em Ovar.

O projeto surge da necessidade de automatização de um processo do setor de Engenharia de Produção da empresa de maneira a que determinadas tarefas se tornem bastante mais rápidas e eficientes. Essa automatização passa pela criação de um conversor de arquivos enviados pelo cliente empresarial, que contém toda a informação de determinada cablagem automóvel, para a linguagem do software utilizado na programação de painéis de testes elétricos das mesmas. O atual processo utilizado despende de bastante tempo, visto que é feito manualmente, pelo que se desenvolveu um sistema em ASP.NET na linguagem C# para otimizar o processo chamado CheckPGM.

Key-words

Electric panels, harness, SQL, ASP.NET, Regular expressions, C#, CheckPGM.

Abstract

This document was written within the final curricular internship of the Master degree on Mathematics and Applications in Computer Sciences, from Aveiro University and its objective is to describe in detail the project developed at the company Yazaki Saltano, in Ovar, Portugal.

This project emerges from the need of automatization of a process in the Production Engineering sector which can become faster and more efficient. This automatization consists of the creation of a converter from the client language, which contains all the information regarding an automotive harness, to the language that a certain internal software uses for electric testing panels programming. Currently, this process takes more than a week to be finished, once it's made manually, therefore it was developed a system in ASP.NET in C# in order to optimize that process named CheckPGM.

LISTA DE ACRÓNIMOS

AJAX – Asynchronous JavaScript And XML.
ASP.NET – Active Server Pages .NET.
CAD – Computer Assisted Design.
CSS – Cascade Style Sheet.
DSI – Design System Interface.
EDT – Editor file.
HTML – Hyper Text Markup Language.
IDE – Integrated Development Environment.
IIS – Internet Information Services.
JS – JavaScript.
MVC – Model-View-Controller.
NYS – New Yazaki System.
PTC – Porto Technical Center.
SGBD – Sistema de Gestão de Base de Dados.
SQL – Structured Query Language.
UML- Unified Modelling Language.

ÍNDICE

1. INTRODUÇÃO 1

1.1. APRESENTAÇÃO DA EMPRESA 1
1.1.1. Setores 3
1.1.2. Formação e Estágio 5
1.1.3. Enquadramento 6
1.2. ELEMENTOS PRINCIPAIS DO PROBLEMA 14
1.2.1. Ficheiro DSI – Design System Interface 14
1.2.2. Ficheiro EDT (Editor) 16
1.3. OBJETIVOS 17

2. MODELAÇÃO DO PROBLEMA E TECNOLOGIAS 19

2.1. MODELAÇÃO 19
2.2. ARQUITETURA 24
2.3. BIBLIOTECAS EXTERNAS AO .NET 28

3. CHECKPGM - A SOLUÇÃO 31

3.1. CAMADA DE NEGÓCIO 31
3.2. Identificação da estrutura do ficheiro DSI 32
3.3. DESENHO DE CABLAGENS 34
3.4. PERMISSÕES 39
3.4. TRATAMENTO DE ERROS 40
3.4.1. Try..catch..finally 40
3.4.2. Programação defensiva 41
3.4.3. Página de erro 41
3.5. INTERFACE 42
3.5.1. ASPETO GERAL 42
3.5.2. MAPA DA APLICAÇÃO 43

4. CONCLUSÕES 55

4.1. KAIZEN 53
4.2. CONSIDERAÇÕES FINAIS 53

REFERÊNCIAS 55

ANEXO A 55

ANEXO B 55

ANEXO C 55

ÍNDICE DE FIGURAS

FIGURA 1 : LOGOTIPO DA EMPRESA 1
FIGURA 2 : ESQUEMA REPRESENTATIVO DA ORGANIZAÇÃO DA EMPRESA 2
FIGURA 3 : ORGANIGRAMA DA EMPRESA EM PORTUGAL, OVAR 2
FIGURA 4 : DIVISÕES DA EMPRESA POR SETOR 3
FIGURA 5 : ESQUEMA REPRESENTATIVO DE UMA CABLAGEM AUTOMÓVEL 4
FIGURA 6 : OUTROS PRODUTOS 4
FIGURA 7: ESQUEMA DE FLUXO DE COMPONENTES 6
FIGURA 8 : RECEPÇÃO DE COMPONENTES 7
FIGURA 9: INSPEÇÃO DE MATERIAIS 7
FIGURA 10: ARMAZÉM DE COMPONENTES 7
FIGURA 11: ORDEM DE FABRICO 8
FIGURA 12: MÁQUINAS DE CORTE E CRAVAÇÃO 8
FIGURA 13: TIPOS DE JUNÇÃO E ACESSÓRIOS 9
FIGURA 14: ÁREA DE SUB-MONTAGEM 9
FIGURA 15: EXEMPLO DE LINHA ROTARY 10
FIGURA 16: EXEMPLO DE LINHA DINÂMICA 10
FIGURA 17: EXEMPLO DE LINHA QE 11
FIGURA 18: EXEMPLO DE LINHA FIXA 11
FIGURA 19: PAINEL DE INSPEÇÃO ELÉTRICA 12
FIGURA 20 YCD, ECRÃ DE TESTE 12
FIGURA 21: MAQUETE DE CLIPS E CLIPS 13
FIGURA 22 : EXEMPLO DE CABLAGEM ELÉTRICA 13
FIGURA 23: EXEMPLO DE ETIQUETA 14
FIGURA 24: ESTRUTURA DE UM FICHEIRO DSI 15
FIGURA 25: ESQUEMA DA GERAÇÃO DO FICHEIRO EDT 16
FIGURA 26: ESTRUTURA DE UM FICHEIRO EDT 16
FIGURA 27: DIAGRAMA DE CASOS DE UTILIZAÇÃO 21
FIGURA 28: MODELO RELACIONAL DA BASE DE DADOS 21
FIGURA 29: MODELO DE ARQUITETURA DE DUAS CAMADAS 26
FIGURA 30: MODELO DE ARQUITETURA DE TRÊS CAMADAS A NÍVEL FÍSICO 26
FIGURA 31: LOGÓTIPO DO MICROSOFT .NET 26
FIGURA 32: EXEMPLO DE DESENHO FEITO COM A BIBLIOTECA WIREIT ADAPTADO 29
FIGURA 33: ESQUEMA DA CAMADA DE NEGÓCIO 31
FIGURA 34: ESQUEMA DE COMUNICAÇÃO ENTRE PARTES DA CAMADA DE NEGÓCIO 30
FIGURA 35: EXEMPLO DE UMA CLASSE GENÉRICA - ACTION 30
FIGURA 36: AUTÓMATO RECONHECEDOR DE UMA LINHA DE SECÇÃO 31
FIGURA 37: AUTÓMATO RECONHECEDOR DE UMA SECÇÃO INTEIRA 32
FIGURA 38: AUTÓMATO RECONHECEDOR DE UMA SECÇÃO VAZIA 34
FIGURA 39: ALGORITMO DA FUNÇÃO GETVIZINHOS 34
FIGURA 40: ALGORITMO DA FUNÇÃO GETCAMINHO 35
FIGURA 41: ALGORITMO DA FUNÇÃO GETNEXTVERTEX 36
FIGURA 42: ALGORITMO DA FUNÇÃO DIJKSTRA 37
FIGURA 43: SINTAXE DAS INSTRUÇÕES TRY,CATCH E FINALLY 37
FIGURA 44: EXEMPLO DE PROGRAMAÇÃO DEFENSIVA: CORREÇÃO DE INPUTS 41
FIGURA 45: LINHA DE CÓDIGO DE ACTIVAÇÃO DA PÁGINA DE ERRO 41
FIGURA 46: ESQUEMA DE PÁGINA 40
FIGURA 47: ESQUEMA DA PÁGINA INICIAL E DE ERRO 40
FIGURA 48: MAPA DA APLICAÇÃO 43
FIGURA 49: MENU INICIAL DA APLICAÇÃO 42
FIGURA 50: MENU FICHEIROS EDT 44
FIGURA 51: PÁGINA DE GERAÇÃO DE FICHEIROS EDT ATRAVÉS DE PARAMETRIZAÇÃO 45
FIGURA 52: PÁGINA DE GERAÇÃO DE FICHEIRO EDT COM PARAMETRIZAÇÃO ÚNICA 46
FIGURA 53: EDITOR DE FICHEIROS EDT 46
FIGURA 54: PÁGINA DE DESENHO DE CABLAGENS 47
FIGURA 55: PÁGINA DE DESENHO – CASO COMPOSIT 47
FIGURA 56: PÁGINA DE DESENHO – CASO MODULAR 48

FIGURA 57: MENU DE CONFIGURAÇÕES 49
FIGURA 58: PÁGINA DE CONFIGURAÇÃO DE PARAMETRIZAÇÕES 49
FIGURA 59: PÁGINA DE EDIÇÃO DA CODIFICAÇÃO DAS CORES 50
FIGURA 60: PÁGINA DE EDIÇÃO DE MARCAS 50
FIGURA 61: PÁGINA DE GESTÃO DE PERMISSÕES 51
FIGURA 62: PÁGINA DE HISTÓRICO 51

ÍNDICE DE TABELAS

TABELA 1: RESUMO DO PROBLEMA 17
TABELA 2: DESCRIÇÃO DO CLIENTE 19
TABELA 3: DESCRIÇÃO DO PRODUTO 20
TABELA 4: RESUMO DAS CAPACIDADES PRETENDIDAS 20
TABELA 5: DESCRIÇÃO PORMENORIZADA DAS CLASSES DO MODELO RELACIONAL 24
TABELA 6: CARATERÍSTICAS DAS LINGUAGENS ORIENTADAS A OBJETOS 35
TABELA 7: CARACTERES DA SINTAXE DAS EXPRESSÕES REGULARES USADOS 33
TABELA 8: TABELA DA CLASSE PONTO 35
TABELA 9: TABELA DA CLASSE LINHA 35
TABELA 10: TABELA DOS UTILIZADORES 40
TABELA 11: KAIZEN 53

Capítulo 1 Introdução

Neste capítulo é feita uma apresentação da empresa onde decorreu o estágio e é feito um enquadramento contextual do funcionamento da mesma para melhor compreensão do problema abordado.

O estágio teve como objetivo o desenvolvimento de uma aplicação web para uso interno da empresa, mais concretamente, no setor de Engenharia de Produção para melhorar a eficiência de um processo que será descrito pormenorizadamente nos capítulos seguintes. Paralelamente ao projeto principal, foram feitas algumas colaborações menores em projetos já existentes.

Este relatório está dividido em quatro capítulos:

Neste capítulo é feita a apresentação da empresa onde decorreu o estágio e uma contextualização do problema.

No Capítulo 2 é feita a análise e modelação do problema, através de descrição de cenários de utilização, modelo de base de dados e toda a planificação que precede a implementação do projeto.

No Capítulo 3 são descritos, em detalhe, alguns pormenores da implementação como algoritmos e técnicas usadas e ainda a descrição do interface com o utilizador.

No último capítulo são apresentadas as conclusões relativas ao estágio.

Existem três anexos a este documento. O Anexo A contém um glossário com alguns termos técnicos específicos; o Anexo B resume a pesquisa realizada sobre ferramentas de desenho de grafos; o Anexo C contém um relatório breve sobre uma aplicação que foi realizada após a conclusão do projeto principal, chamada VoIP que faz uma gestão dos contactos do sistema VoIP.

1.1. Apresentação da empresa

A Yazaki (logótipo na figura 1) é uma empresa cujas raízes são oriundas do Japão. Foi fundada em 1929 por Sadami Yazaki e começou a sua atividade através da produção de fios elétricos para automóveis, abrangendo, a partir 1935, a produção para automóveis domésticos. Com a evolução da indústria automóvel presente na altura, foi, em 1949, tomada a decisão de concentrar a atividade da empresa na produção de cablagens automóveis, passo que foi decisivo para o crescimento da empresa[21].

Figura 1 : Logotipo da empresa

Atualmente o grupo Yazaki tem inúmeras localizações espalhadas pelo mundo inteiro [21], como se pode verificar na imagem seguinte, que incorporam áreas de investigação e desenvolvimento, produção, vendas e gerência local.

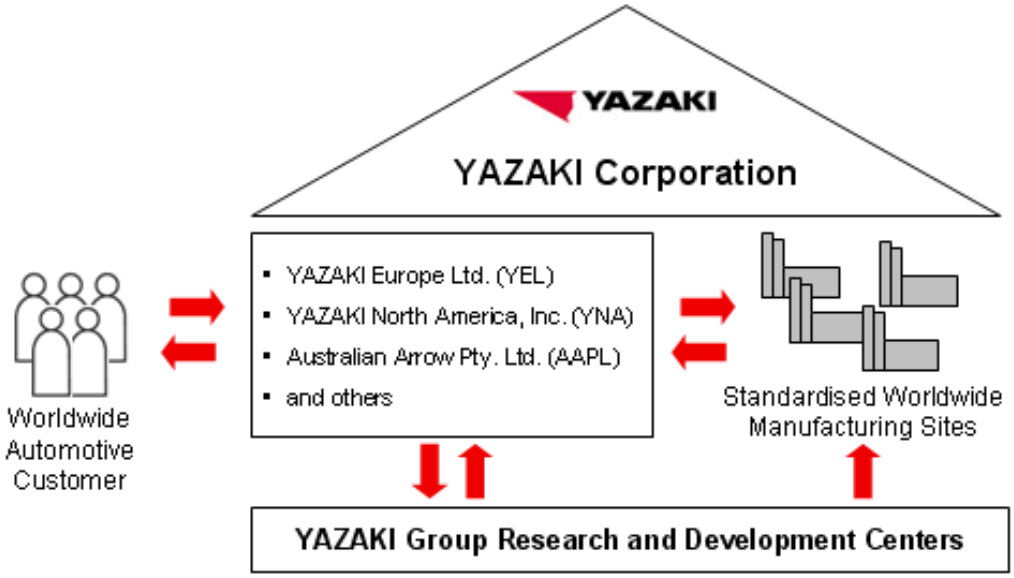


Figura 2 : Esquema representativo da organização da empresa

A Yazaki em Portugal tem colaboradores distribuídos pelas áreas: Produção de Cablagens, Produção de Componentes e pelo PTC (*Porto Technical Center* – Desenho Técnico de Cablagens e Laboratório) e é organizada de acordo com o seguinte organigrama[5]:

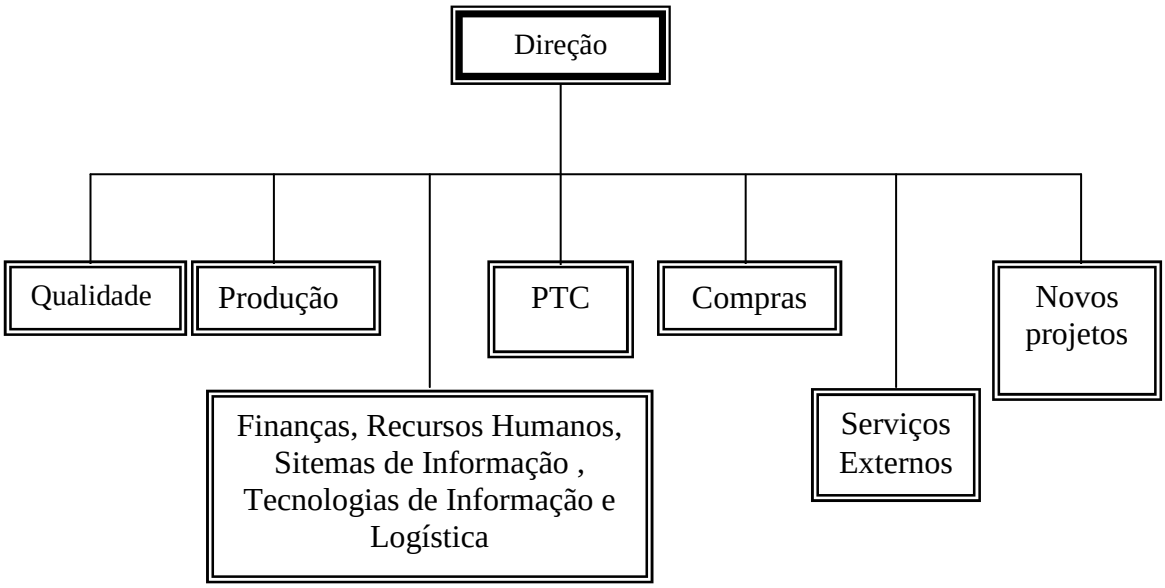


Figura 3 : Organigrama da empresa em Portugal, Ovar

1.1.1. Setores

O grupo Yazaki fornece produtos para a área Automóvel e Sistemas Ambientais. Atualmente expandiu-se para um terceiro setor englobando cuidados de enfermagem e negócios relacionados com o ambiente como demonstra o esquema da figura 4[21].

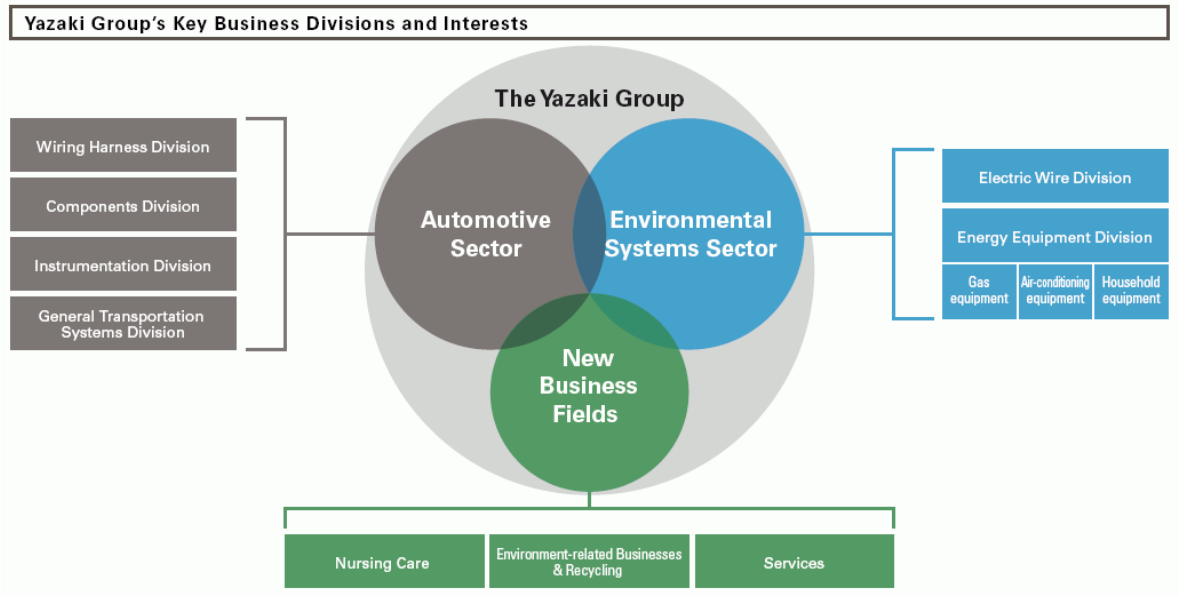


Figura 4 : Divisões da empresa por setor

- Setor Ambiental

As competências da Yazaki desenvolvidas ao longo do tempo, foram usadas na criação de vários tipos de equipamentos e produtos que suportam o fornecimento e utilização de várias fontes de energia, como cabos de transmissão elétrica, sistemas de segurança de gás, sistemas de ar-condicionado, entre outros. Consequentemente, a Yazaki introduziu-se numa sociedade amiga do ambiente, aplicando os seus princípios nas operações ambientais e energéticas, sendo o segundo maior setor do grupo Yazaki[21].

- Setor automóvel

A Yazaki é fornecedora de um largo espetro de produtos que se aplicam à eletrónica automóvel, cablagens elétricas (figura 5)[5], instrumentação e sub-montagens eletrónicas. A linha de produtos inclui fibra ótica, mostradores e módulos de relógios, centrais de energia, componentes de eletrónica, interruptores, conetores, terminais e cabos de alta voltagem (figura 6)[5]. Cerca de 90% do negócio deriva deste setor e será este ao qual o projeto se aplica.

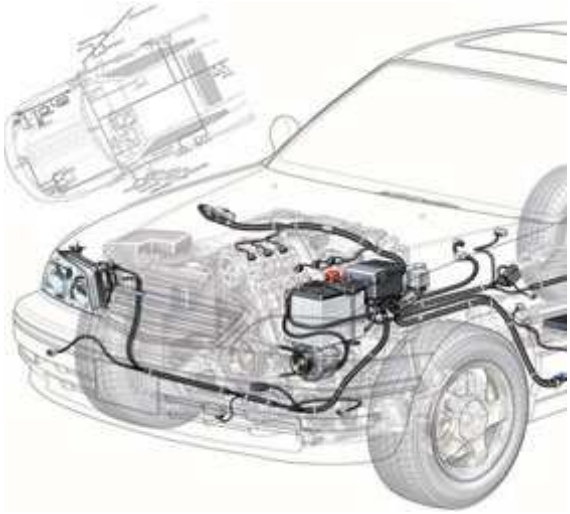


Figura 5 : Esquema representativo de uma cablagem automóvel

Figura 6 : Outros produtos: em cima – fio; em baixo da esquerda para a direita: Conetores, painéis de mostradores, fibra ótica e desenho técnico

1.1.2. Formação e Estágio

O estágio teve a duração de 7 meses e foi constituído por duas fases: uma primeira (entre 3 de outubro e 31 de dezembro de 2011) em que apenas englobava dois dias por semana, segunda e terça-feira; e posteriormente (até 31 de maio de 2012) em horário completo das 08:00h às 17:00h.

Todos os novos colaboradores recebem uma formação inicial para permitir uma melhor perceção das políticas e metodologias da empresa, bem como fornecer um enquadramento da área de negócio em que se vai inserir. No caso da autora da dissertação, o plano de formação teve duas fases, tendo sido a primeira semana focada nas formações teóricas nas áreas que se seguem:

- Apresentação da empresa.
- Ambiente: consiste na apresentação da certificação ambiental da empresa e dos métodos de reciclagem existentes.
- Higiene, Saúde e Segurança no Trabalho.
- Fluxo e Componentes: aqui foram introduzidos os conceitos das componentes usadas na produção de cablagens e o percurso das mesmas na linha de produção.
- "Muda": em japonês significa desperdício, e como a palavra indica, esta é uma política que pretende eliminar os desperdícios.
- *New Yazaki System* (NYS): tem como objetivo otimizar a capacidade de reação às mudanças por parte dos clientes.
- Qualidade: técnicas de verificação de qualidade das cablagens e os seus componentes.
- *Total Productive Maintenance on Equipment* (TPME): metodologia que visa uma manutenção autónoma do sistema de produção por parte dos colaboradores, de maneira a evitar avarias nos equipamentos.

A segunda semana incidiu sobre formação prática onde a autora teve oportunidade de acompanhar alguns dos engenheiros de produção no seu dia de trabalho, para se familiarizar com o contexto em que o projeto se enquadra.

Na terceira semana já foi integrada no departamento de Sistemas de Informação e desenvolvimento, onde permaneceu até ao fim do estágio.

1.1.3. Enquadramento

Cada produto passa por um conjunto de fases distintas, desde que as suas componentes chegam à fábrica até serem expedidos. Um conjunto de componentes passa por três fases distintas: receção, montagem e expedição. Cada uma destas fases divide-se noutras, que serão descritas com maior pormenor de seguida. O esquema da figura 7 mostra o percurso que cada produto e as suas componentes percorrem.

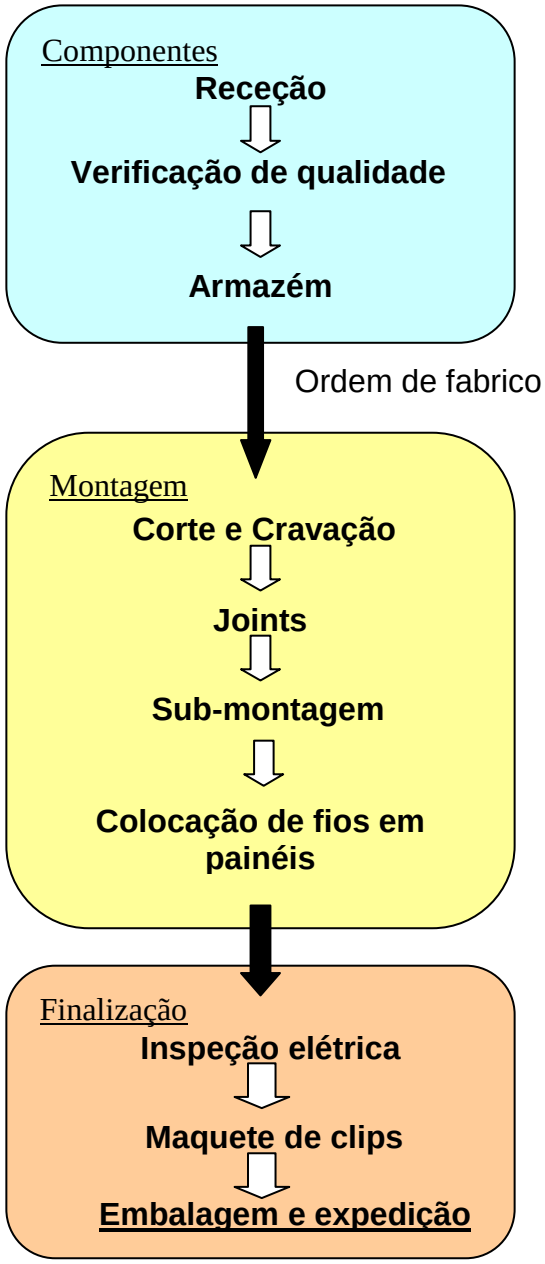


Figura 7: Esquema de fluxo de componentes

As figuras seguintes ilustram as várias etapas do esquema anterior.

- **Receção**

Os materiais que chegam à Yazaki são conduzidos para a área de receção (figura 8) onde são codificados.

Figura 8 : Receção de componentes

- **Inspeção de materiais**

Os componentes são inspecionados no gabinete de qualidade (figura 9) de acordo com os desenhos e especificações aplicadas. Se o material estiver correto é colocado nas embalagens uma etiqueta de OK, caso contrário é retirado e posto em local apropriado para o efeito[7].

Figura 9: Inspeção de materiais

- **Armazém**

Terminada a inspeção, os materiais são transferidos para o armazém de stock (figura 10) sendo posteriormente fornecidos à produção.

Figura 10: Armazém de componentes

- **Ordem de fabrico**

É um bilhete de identidade (ver figura 11) de um circuito. Contém a seguinte informação: espécie, secção, cor e comprimento do fio, nº da máquina, nº do operador, código dos terminais a cravar, comprimento da desfolha e códigos de acessórios (componentes a adicionar ao circuito)[7]. Esta informação é gerada pelo setor engenharia e aplicada à cablagem em causa.

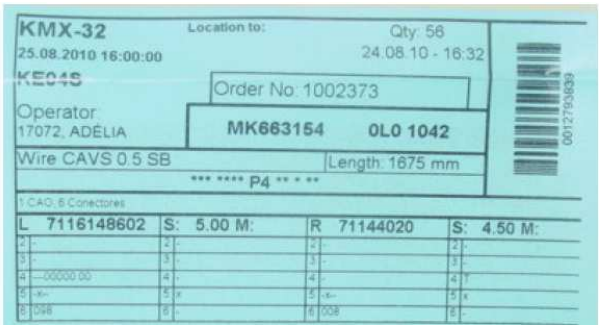


Figura 11: Ordem de fabrico

- **Corte e Cravação**

O corte e cravação de fios com as peças terminais podem ser feitos automaticamente ou manualmente. Existem várias máquinas para a cravação automática, como se vê na figura seguinte.

Figura 12: Máquinas de corte e cravação

- **Junções (Joints)**

Após o corte e cravação os fios passam para os *joints* (junções de fios), caso sejam necessários. Pode ser feito através e cravação, solda (através de corrente elétrica) ou através de ultrasons (ver figura 13, esquerda). É também feita uma solda nos terminais, quando necessário. Os fios passam para a mesa de acessórios onde são efetuadas operações adicionais (colocação de bucha, coto, entre outros - ver figura 13, direita) e passam para a área de distribuição final: os circuitos são distribuídos por produto em cavaletes que têm uma lista de verificação onde são registados os circuitos que vão chegando à distribuição. Os cavaletes podem estar identificados por linha, estação ou nº

de circuito. A lista de verificação é também produzida pela engenharia de modo a que se saiba a que produto pertence cada conjunto de fios[7].

Figura 13: Tipos de junção (à esquerda) e acessórios (à direita)

- **Processo de sub-montagem**

Da zona de distribuição, os fios passam para a sub-montagem: processo manual no qual um conjunto de fios é inserido no conetor respeitando um desenho de sub-montagem (ver figura 14, esquerda). Pode ser colocado antes da inserção de vários tipos de acessórios, como se pode ver na figura seguinte, do lado direito[7]. Aqui existem ajudas visuais (por exemplo, etiquetas identificativas das cores nas caixas dos conetores) que são feitas pela engenharia de produção, bem como os desenhos de sub-montagem.

Figura 14: Área de sub-montagem

- **Colocação de fios**

Os colaboradores distribuídos ao longo da linha de montagem (figuras 15,16 e 17) vão colocando as sub-montagens no painel (ver figura 18). No painel está devidamente identificada a localização e desenho dos conetores. Aqui também podem ser inseridos alguns fios no conetor, completando a ligação de todos os circuitos e podem ser adicionados isolamentos com fitas ou vinil. Todas as ajudas visuais, normas de

manuseamento e desenhos dos painéis estão ao cargo do departamento de engenharia de produção[7]. Existem vários tipos de linhas de montagem, entre elas a linha *Rotary*, a linha Dinânima, a linha QE e a linha fixa que passamos a descrever.

Linha *Rotary*

Tem como função permitir operações de montagem em linha tipo carrossel, como se mostra na figura seguinte. A linha é constituída por vários painéis que deslizam sobre calhas. A linha pode estar sempre em movimento ou apenas movimentar-se dentro de um período de tempo fixo e permite que a cablagem seja transportada entre postos de trabalho.

Figura 15: Exemplo de linha Rotary

Linha Dinâmica

Permite operações de montagem em linha, com postos de trabalho fixos. Para tal, dispõe de carros que se deslocam sobre guias e realiza o transporte entre os postos de trabalho[18]. Dispõe também de um elevador e de um túnel inferior para transportar os painéis virados para cima. Este tipo de linha distingue-se da anterior pela dimensão dos painéis, que são consideravelmente mais pequenos (ver figura 16) em relação aos da linha *Rotary*.

Figura 16: Exemplo de linha Dinâmica

Linha QE

Permite operações de montagem em linha, em postos de trabalho fixo (figura 17). A linha dispõe de um tapete rolante que transporta a o produto entre postos de trabalho[18].

Figura 17: Exemplo de linha QE

Linhas Fixas

São linhas com painéis fixos, onde não há qualquer movimento automático das cablagens, como se pode verificar na figura 18. São usadas para cablagens que são mais específicas ou não são produzidas em grandes quantidades.

Figura 18: Exemplo de linha fixa

- **Inspeção elétrica**

Após a montagem da cablagem verifica-se a continuidade elétrica dos circuitos, a correta posição dos mesmos e a presença de componentes em painéis próprios (ver figura 19). Podem também ser colocados díodos, fusíveis, etc. Este processo tem uma importância acrescida visto que é a barreira principal aos circuitos erróneos e à falta de algum componente[7]. É sobre esta etapa que o projeto irá incidir.

Figura 19: Painel de inspeção elétrica

Para a verificação das ligações ser feita, é usado um software próprio, que indica todas as conexões e deteções a testar (como por exemplo, detetar a presença de uma peça). Este software emite avisos visuais para indicar ao colaborador que terá de fazer a tarefa indicada no ecrã (figura 20). Atualmente, estão a ser utilizados vários softwares externos, no entanto a sua utilização irá ser descontinuada e já está a ser introduzida a utilização de um software interno da empresa chamado YC-D (Yazaki Checker, versão D). Este software tem um tipo de ficheiros de *input* próprios (ficheiros EDT), que irão ser descritos em pormenor posteriormente.

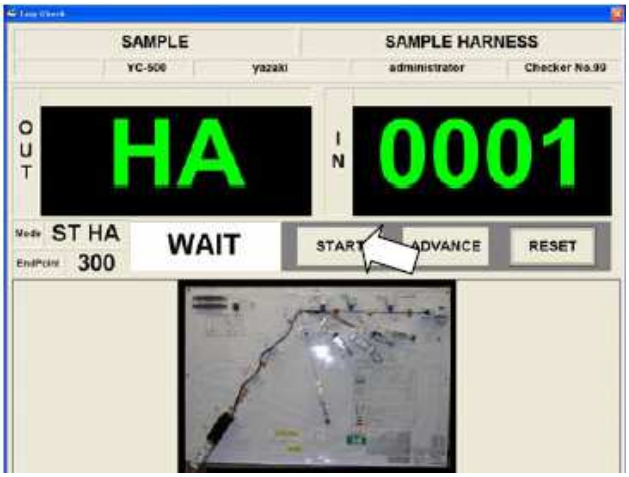


Figura 20 – YCD, ecrã de teste

- **Maquete de *clips***

Os *clips* (ver figura seguinte, direita) são uma parte importante da cablagem pois têm como função segurar a mesma ao chassis do automóvel. Estes têm posições específicas que devem ser verificadas. Quando se torna necessário, ou seja, quando existe um número de clips que assim o exige, a cablagem passa por uma maquete de clips (figura 21, esquerda) que verifica a colocação dos mesmos através de sensores[7].

Figura 21: Maquete de clips (esquerda) e *clips* (direita)

- **Segunda inspeção visual**

Em alguns casos a cablagem passa por uma segunda inspeção visual onde a mesma é comparada com o esquema de um painel. São verificadas as medidas, posição dos ramos e cruzamentos, terminais, protetores, clips, etc. O próximo passo é a embalagem do produto final (figura 22) e expedição deste.

Figura 22 : Exemplo de cablagem elétrica

- **Embalagem e Expedição**

Caso a cablagem passe na inspeção elétrica e na inspeção visual, caso se aplique, é emitida uma etiqueta identificativa com um código de barras contendo o número identificativo, nível de engenharia, data, linha, quantidade, entre outras informações (figura 23). Esta informação fica também registada num software de gestão de encomendas. Este passo é dos mais importantes, porque tem como objetivo verificar se as cablagens embaladas estão a ser inseridas na embalagem correta, na quantidade certa[7]. Mais tarde será verificado esse mesmo código de barras da embalagem com a informação dada pelo sistema para uma nova verificação de todos os produtos a enviar ao cliente.

Figura 23: Exemplo de etiqueta

1.2. Elementos principais do problema

Como já foi dito anteriormente, o projeto aplica-se ao setor de montagem de cablagens automóveis, mais concretamente ao setor de Engenharia de Produção da empresa. O setor de Engenharia está encarregue da definição de processos e do planeamento de cada uma das fases da construção de cada cablagem. Após esta estar montada, é necessário proceder a um teste de verificação de continuidade elétrica entre os pontos da cablagem, que é feito usando software interno da empresa. Este software recebe como entrada um ficheiro de texto, de extensão EDT - Editor) contendo informação extraída do ficheiro enviado por clientes da empresa, o ficheiro DSI, que se descreve seguidamente.

1.2.1. Ficheiro DSI – *Design System Interface*

Trata-se de um ficheiro de texto exportado de ferramentas CAD (Computer-assisted-design) que contém informação sobre determinada cablagem ou conjunto de cablagens. Este ficheiro contém várias secções diferentes, sendo que cada secção contém um conjunto de linhas correspondentes que por sua vez contém campos com a informação. Pode ser lido como qualquer ficheiro de texto e tem uma estrutura padrão bem definida através de símbolos especiais, como se observa na figura seguinte:

```
! Comentário
% Nome da secção 1
campo1:campo2:campo3:campo4
campo1:campo2:campo3:campo4
campo1:campo2:campo3:campo4
campo1:campo2:campo3:campo4
% Nome da secção 2
campo1:campo2:campo3:campo4:campo5:campo6
campo1:campo2:campo3:campo4:campo5:campo6
campo1:campo2:campo3:campo4:campo5:campo6
campo1:campo2:campo3:campo4:campo5:campo6
```

Figura 24: Estrutura de um ficheiro DSI

A cada símbolo que aparece está associado um significado que define a estrutura do ficheiro:

% - Delimitador de secção.
! - Comentário.
: - Delimitador de campos de cada linha.

O ficheiro DSI pode aparecer em três variantes diferentes, todas baseadas na estrutura padrão definida anteriormente:

- *Composite*: é uma variante baseada no modelo anterior, que descreve um conjunto de sub-produtos (chamadas derivativas) e as respectivas opções associadas. Pode-se, portanto, pretender extrair a informação de uma só derivativa que corresponde a uma só cablagem com diferentes opções pré-definidas.

- *Modular*: esta variante tem a informação diferenciada por módulos, também definidos em secção própria. Esta variante tem a vantagem de se poder construir cada produto conjugando módulos diferentes. Para extrair a informação do produto pretendido é necessário indicar quais módulos que lhe pertencem. Esta indicação não está expressa no ficheiro DSI e terá de ser dada pela pessoa que o manipula. A diferença relativamente à variante composite reside no fato de os módulos poderem ser escolhidos, caso *modular*, enquanto que na *composite* o conjunto de escolhas é fixo.

- *Single*: variante mais simples, que não contém qualquer referência a módulos ou derivativas, representando assim a informação de uma única cablagem.

Como o esquema da figura seguinte mostra, um cliente empresarial pode enviar o ficheiro DSI ou o desenho técnico para o PTC. Caso seja enviado o desenho técnico este é exportado para ficheiro DSI previamente, depois o ficheiro é enviado para o setor de Engenharia. Posteriormente, um Engenheiro de Produção irá tratar o ficheiro manualmente criando o chamado ficheiro EDT (Editor), que será descrito em seguida.

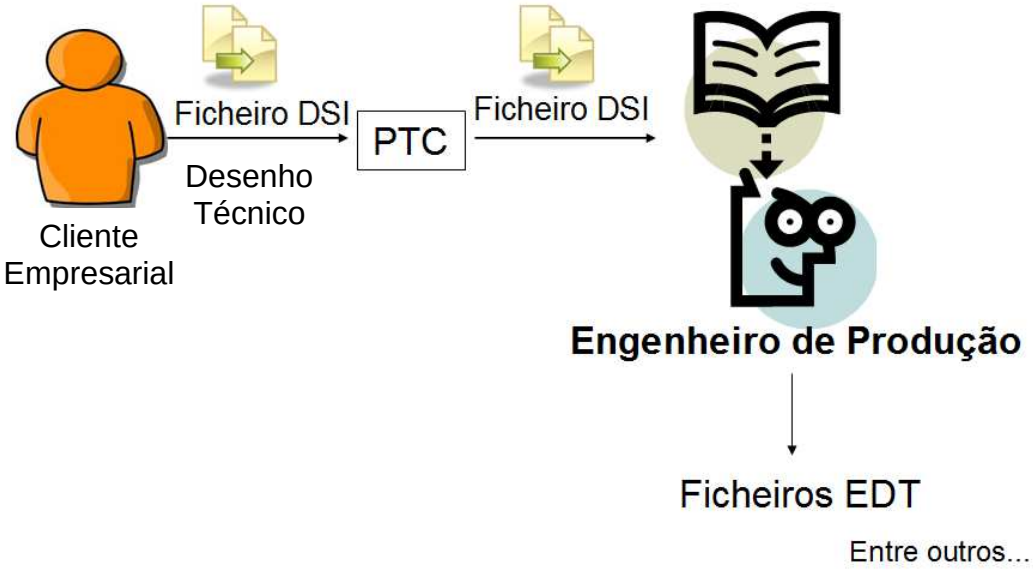


Figura 25: Esquema da geração do ficheiro EDT

1.2.2. Ficheiro EDT (Editor)

O ficheiro DSI tem diferentes propósitos, sendo que no caso particular do projeto, tem como objetivo fornecer a informação necessária para gerar o ficheiro de *input* destinado ao software interno da empresa que se encarrega dos testes elétricos efetuados a cada cablagem. Esses novos ficheiros têm a extensão .EDT e podem também ser abertos em qualquer editor texto, bem como em folha de cálculo, ou no software interno para edição. Tem uma estrutura simples, em que cada linha contém um conjunto de campos extraídos (por exemplo: campo de cores, campo de nome de fio, etc.) de determinada secção do ficheiro DSI, delimitados por uma vírgula, conforme mostra a figura 26:

```
nome1,cavidade1,nome2,cavidade2,cor1,cor2,cor3,valor,valor,mensagem
nome1,cavidade1,nome2,cavidade2,cor1,cor2,cor3,valor,valor,mensagem
nome1,cavidade1,nome2,cavidade2,cor1,cor2,cor3,valor,valor,mensagem
nome1,cavidade1,nome2,cavidade2,cor1,cor2,cor3,valor,valor,mensagem
nome1,cavidade1,nome2,cavidade2,cor1,cor2,cor3,valor,valor,mensagem
```

Figura 26: Estrutura de um ficheiro EDT

Nenhum dos campos referidos na figura 26 sofre alterações, ou seja, a informação referente a cada campo é copiada diretamente do ficheiro DSI, com exceção dos campos referentes a cores. Existem três campos destinados a cores (cor1,cor2,cor3), cores essas que aparecem codificadas no ficheiro DSI, como por exemplo, "BN" corresponde a castanho (*brown*), "RD" corresponde a vermelho (*Red*). Torna-se então importante identificar cada cor corretamente, bem como registar diferenças entre marcas (por exemplo: "P" pode corresponder à cor Preto em português, e a *Pink* em inglês).

1.3. Objetivos

Como já foi dito, o setor de Engenharia de Produção recebe a informação dos circuitos elétricos das cablagens a serem produzidas. Após receber esta informação procede à sua descodificação de forma a configurar os painéis de inspeção elétrica. O tratamento da informação proveniente do ficheiro DSI e posterior criação do ficheiro EDT é feito manualmente e varia consoante a marca em causa. A tabela seguinte faz um resumo do problema a tratar, qual a sua importância e qual a solução pretendida.

Os problemas	O tempo gasto na criação do *input* e as diferenças na informação recebida de marcas diferentes.
Afeta	Engenheiros e Colaboradores.
E que tem impacto em...	Provoca atrasos e é mais propenso a erros humanos.
Uma solução bem sucedida seria...	Sistematizar todos estes processos de maneira a que unifique todas as marcas e a que seja bastante mais rápido (reduzir para 1 dia) e mais fiável.

Tabela 1: Resumo do problema

O principal objetivo deste projeto consiste na implementação de um sistema de informação que permita converter a informação obtida a partir de um ficheiro DSI num ficheiro do tipo EDT, ou seja, pretende-se elaborar um conversor automático capaz de descodificar a informação enviada pelo cliente empresarial e, segundo determinado protocolo, produza automaticamente o *input* para a configuração dos painéis, para reduzir o tempo despendido para o efeito. A aplicação desenvolvida terá de ser de fácil utilização, ter um aspeto profissional e deverá preencher todos os requisitos necessários à tarefa em causa. É necessário também que a aplicação tenha em conta que os dados manuseados são confidenciais e portanto terá de contemplar um mecanismo de segurança para proteção dos mesmos.

Capítulo 2 Modelação do problema e Tecnologias

A solução para o problema apresentado em 1.3 passa por desenvolver uma aplicação que seja capaz de reconhecer a estrutura do ficheiro DSI, que extraia a informação necessária para gerar o ficheiro EDT e fornecer um adequado interface gráfico. Este capítulo tem como objetivo descrever a modelação realizada para o problema, bem como as tecnologias que serviram de ferramentas para a implementação do sistema.

2.1. Modelação

Para a modelação deste problema são utilizadas técnicas de *Unified Modelling Language* – UML versão 2.0. Nesse sentido, são descritas as necessidades e requisitos do sistema e são apresentados vários tipos de diagramas. Em termos de diagramas de estruturação, criou-se um diagrama de classes e em termos de diagramas descritivos de comportamentos temos um diagrama de casos de utilização que resume as funcionalidades existentes e as associa aos seus atores. Também é descrita a arquitetura escolhida para o sistema, entre outras informações essenciais.

Cliente

Entende-se por cliente o utilizador final da aplicação. É aquele que solicitou a aplicação e que determina quais os requisitos e características que a mesma vai englobar. Na tabela seguinte faz-se uma análise do cliente final do projeto: o Departamento de Engenharia de Produção da empresa.

Cliente	Descrição	Responsabilidades
Departamento de Engenharia de Produção	Engenheiros encarregues da programação das consolas de inspeção elétrica.	O departamento de Engenharia de Produção tem como principal objetivo implementar todos os mecanismos necessários à produção, mais concretamente, neste caso, está responsável pela montagem e programação de todos os painéis de teste.

Tabela 2: Descrição do cliente

O produto

Produto é a designação atribuída à aplicação desenvolvida em si. É portanto a solução produzida para o problema descrito em 1.3. Na tabela 3 resume-se o conjunto de características do produto final.

Para	Uso no Departamento de Engenharia de Produção
O que é	Um conversor automático de linguagem do cliente para liguagem de configuração de painéis e desenho dos mesmos.
Nome do produto	CheckPGM
Que vai	Facilitar o processo de configuração de painéis de inspeção elétrica, reduzindo-o para 1 dia.
Ao contrário do que existe	Atualmente demora cerca de uma semana.

Tabela 3: Descrição do produto

Estratégia da solução

Como foi referido em 1.2.1, um ficheiro DSI contém várias secções diferentes assinaladas com o símbolo "%" e cada uma delas contém um conjunto de linhas com campos delimitados pelo símbolo ":" (ver figura 24). Para poder extrair a informação certa é necessário saber a sua localização no ficheiro original, mais concretamente, a secção em que está contida e a posição em cada uma das linhas. Assim criou-se o conceito de parametrizações de cada cliente empresarial ou marca de automóveis, em que para cada campo necessário (por exemplo: campo das cores) se regista a posição na linha (por exemplo: campo nº4 de cada linha) e a secção correspondente. É necessário então:

- Registar a parametrização de cada campo necessário no ficheiro DSI.
- Registar a codificação de cores como foi referido em 1.2.2, ou seja, registar o código e o nome de cada cor (por exemplo, BN corresponde a *Brown*).
- Registar cada marca e associar-lhe um conjunto de parametrizações e uma codificação de cores única.

Resumo das capacidades pretendidas

Existe um conjunto de capacidades genéricas que o sistema contempla, capacidades essas resumidas na tabela seguinte.

Benefícios	Característica correspondente do sistema
Tempo Despendido	Tarefas automatizadas
Correção de inputs	Linguagens Regulares
Maior facilidade na mudança de marca	Registo de todas as marcas e parâmetros correspondentes
Sistema eficiente	Tarefas automatizadas
Fácil utilização	Interface intuitivo e simples

Tabela 4: resumo das capacidades pretendidas

Cenários de utilização

- **Atores**

O sistema desenvolvido tem dois tipos de atores (utilizadores): o utilizador genérico e o administrador. O administrador herda as permissões do utilizador e pode ainda modificar dados de registos (histórico e permissões). O mecanismo de reconhecimento de permissões será descrito posteriormente.

- **Funcionalidades**

Para modelar devidamente um sistema, torna-se necessário definir exatamente quais as funcionalidades que o mesmo deverá incorporar, bem como definir quais os atores que lhes têm acesso e como. A figura seguinte visa esquematizar estas mesmas funcionalidades e relacioná-las com o ator que pode fazer uso delas.

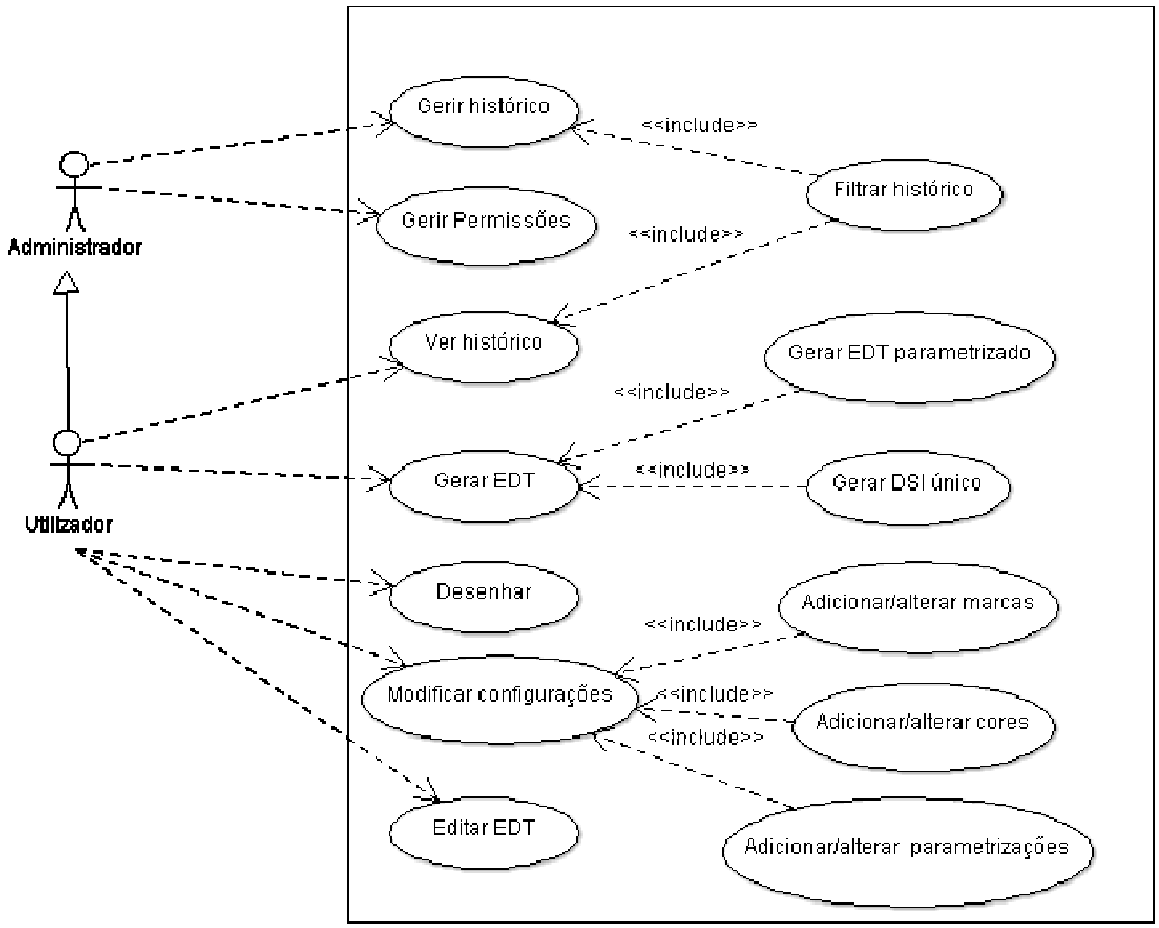


Figura 27: Diagrama de casos de utilização

Resumidamente e de acordo com a figura anterior, a aplicação contém as seguintes funções:

- Gerar ficheiro EDT a partir de parametrizações guardadas em base de dados.

- Gerar ficheiro EDT a partir de parametrizações únicas, para casos em que surjam parametrizações não registadas em base de dados.

- Editar ficheiro EDT já existente.

- Alterar/Criar parametrizações/cores/marcas/permissões.

- Desenhar cablagem: funcionalidade extra.

- Ver histórico.

- Alterar histórico (administrador).

Modelo do domínio

Para guardar as parametrizações de cada marca criou-se um modelo de dados relacional que está representado na figura seguinte:

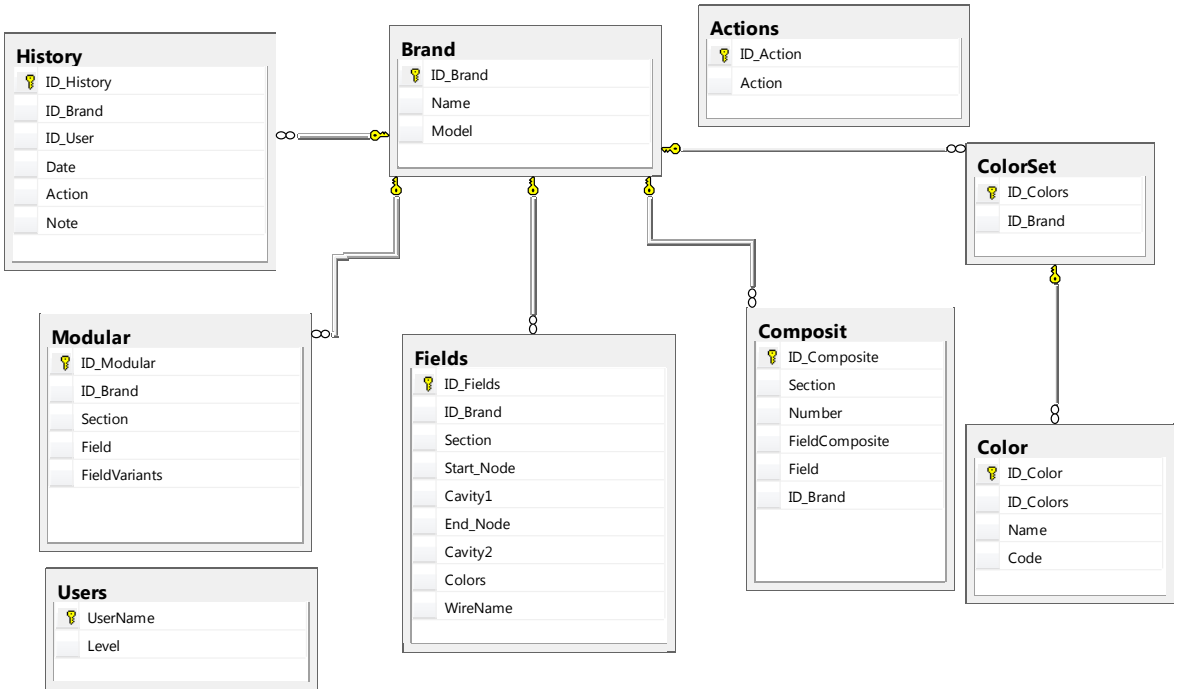


Figura 28: Modelo relacional da base de dados

Descrição do modelo relacional

De acordo com a figura 28, existem as seguintes tabelas e correspondentes atributos:

Conceito	Atributo	Descrição[1]
Brand	ID_Brand	ID da marca
	Name	Nome da marca
	Model	Modelo, se aplicável
History	ID_History	ID do histórico
	ID_Brand	ID que identifica a marca
	ID_User	ID do utilizador em causa
	Date	Data da ação
	Action	Descritivo da ação
	Note	Nota opcional
ColorSet	ID_Colors	ID do esquema de cores
	ID_Brand	ID que identifica a marca
Color	ID_Color	ID da cor
	ID_Colors	ID do esquema de cores
	Name	Nome da cor
	Code	Código da cor
Fields	ID_Fields	ID dos campos
	ID_Brand	ID que identifica a marca
	Start_node	Nó inicial
	Cavity1	Cavidade do nó inicial
	End_Node	Nó final
	Cavity2	Cavidade do nó final
	Colors	Campo das cores
	Equipotential	Campo do equipotencial
Actions	ID_Action	ID da ação
	Action	Descrição da ação
Composite	ID_Composite	ID do registo
	Section	Nome da secção
	Number	Campo com o produto
	FieldComposite	Campo das opções nos fios
	Field	Campo com as opções
	ID_Brand	ID que identifica a marca
Modular	ID_Modular	ID do registo
	ID_Brand	ID que identifica a marca
	Section	Nome da secção
	Field	Campo com módulo
	FieldVariants	Campo do módulo nos fios
Users	UserName	Nome de utilizador
	Level	Nível de permissão

[1] Por ID entende-se identificador unívoco através de um valor inteiro atribuído automaticamente.

Tabela 5: Descrição pormenorizada das classes do modelo relacional

- **Brand:** Classe identificativa da marca.

- **History:** Classe cujos atributos documentam todas as ações realizadas na aplicação.

- **ColorSet:** Classe que identifica uma codificação de cores.

- **Color:** Classe que contém o nome e o correspondente código.

- **Users:** Classe que identifica um utilizador e lhe associa um nível.

- **Fields:** Classe que identifica a localização no ficheiro DSI de cada campo necessário para o ficheiro EDT.

- **Actions:** Classe que contém todas as ações possíveis a serem utilizadas no campo “Action” da tabela “History”.

- **Composite:** Classe que contém todas as parametrizações da informação de ficheiros composite.

- **Modular:** Classe que contém todas as parametrizações da informação de ficheiros modular.

2.2. Arquitetura

O sistema desenvolvido é uma aplicação que vai integrar a intranet da empresa. Uma intranet é um conjunto de serviços da internos a uma rede local, ou seja, acessíveis a partir de diferentes máquinas da rede, ou a partir um conjunto de redes bem definidas e invisíveis do exterior. Consiste, portanto, em utilizar um modelo cliente-servidor para criar um sistema de informação interno a uma empresa[9]. Existem várias arquiteturas que podem cumprir estes requisitos e existem dois modelos que se podem considerar os mais comuns: modelos em duas e três camadas.

2.7.1. Modelo em duas camadas

Aqui apenas existe a camada de dados e a camada de cliente (figura 29). Os pedidos são feitos diretamente à base de dados, pelo que a camada de cliente está encarregue de aplicar todas as regras de negócio (validações, tratamento de exceções, regras que o sistema deverá cumprir), pelo que o código correspondente terá de estar contido em todas as máquinas[16].

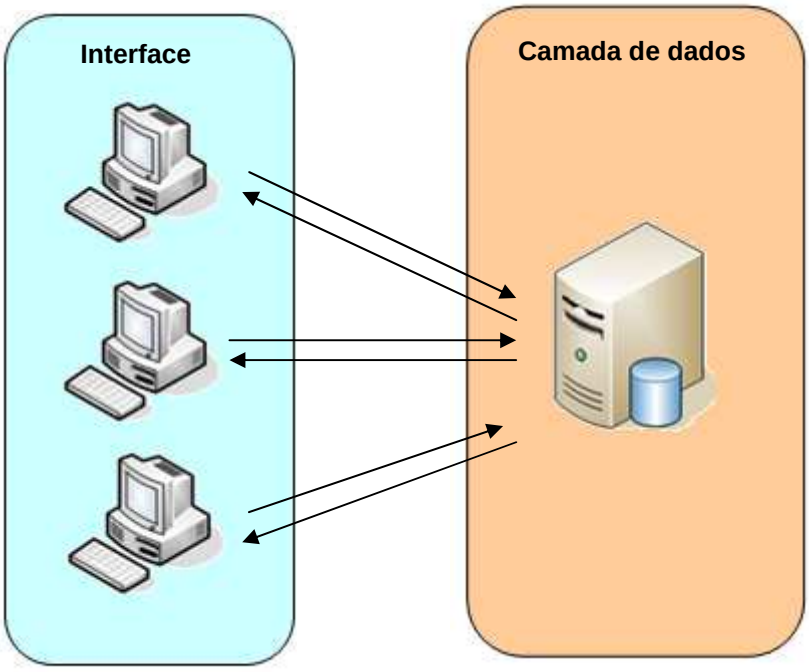


Figura 29: Modelo de arquitetura de duas camadas

2.7.2. Modelo em três camadas

O modelo em três camadas oferece uma melhoria em relação ao modelo de duas camadas, quando se pretende que haja uma maior distinção entre as regras de negócio e a camada de interface conseguindo assim uma maior portabilidade das mesmas. Também introduz um segundo servidor intermédio, pelo que diminui os requisitos da máquina cliente. É constituída portanto por três camadas[3], como se verifica na figura 30:

- Camada de interface

  A aplicação web, responsável pela interface com o utilizador através de um navegador de internet (Internet Explorer, Firefox, Chrome, etc). implementada através das componentes visuais da plataforma .NET. O objetivo é permitir ao programador obter produtividade através da facilidade do desenvolvimento interface. As classes dessa camada utilizam os serviços oferecidos pela camada de negócios[16].

- Camada de negócio (*Business Tier*)

  Um ou vários servidores de aplicação: permite tratar de questões de segurança, tratamento de exceções, comunicar com a camada de dados através de pedidos SQL[2]. Serve portanto de ponte entre a camada de interface e a camada de dados, e protege os mesmos, uma vez que o interface não tem acesso direto aos dados[14]. Aqui estão contidas todas as regras de negócio, contrariamente ao modelo de duas camadas.

- Camada de dados: É responsável pela persistência e acesso aos dados da aplicação[12]. Esta camada recebe os pedidos da camada de negócio e os seus métodos alteram as tabelas da base de dados[19].

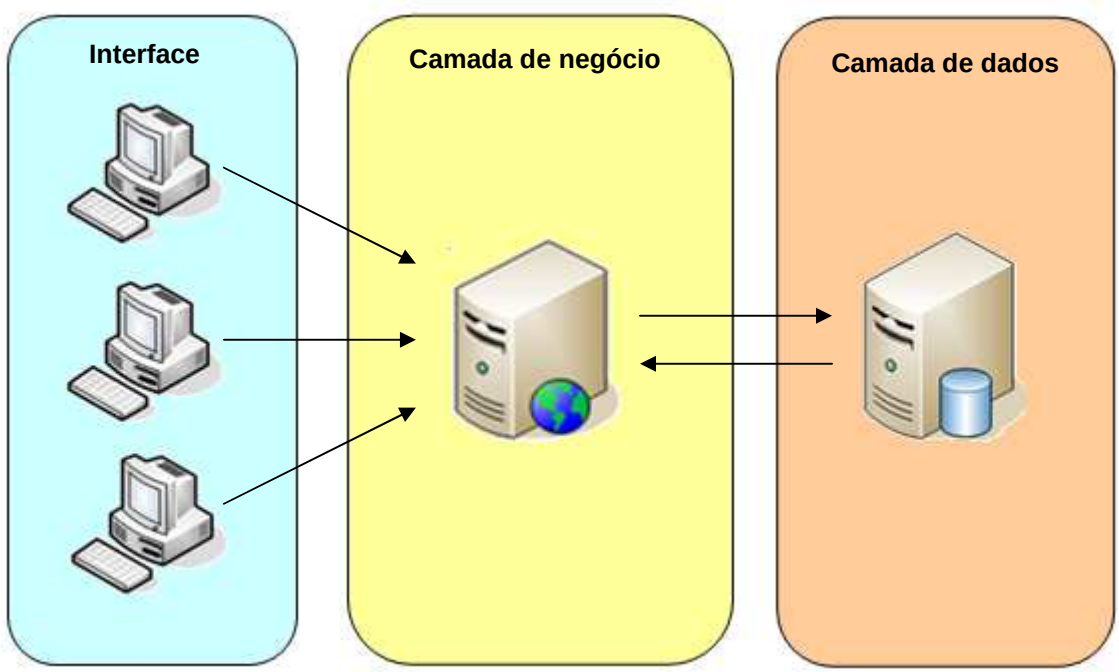


Figura 30: Modelo de arquitetura de três camadas

O modelo escolhido para esta aplicação foi o de três camadas visto ser necessário ter em conta a segurança dos dados e a performance da aplicação.

2.3. Ferramentas e linguagens utilizadas

No decorrer do projeto, o volume de ferramentas usadas foi aumentando consideravelmente, pelo que achou-se pertinente referir brevemente cada uma delas.

- *Framework* .NET

O .NET *Framework* (figura 31) é uma plataforma de software desenvolvida pela Microsoft. Contém uma grande variedade de bibliotecas e uma interoperabilidade entre linguagens, ou seja, cada linguagem pode utilizar bibliotecas escritas noutras linguagens[15]. A biblioteca base da plataforma permite criar o interface com o utilizador, acesso a dados, conexões a bases de dados, criptografia, desenvolvimento de aplicações web, algoritmos numéricos e comunicações de rede. Cada programador produz software combinando o seu próprio código com o da plataforma .NET e outras bibliotecas[15]. O principal ambiente de programação (IDE- *Integrated Development Environment*) para o .NET é chamado Visual Studio e é o utilizado neste projeto, com a linguagem C#.

Figura 31: Logótipo do Microsoft .NET

- ASP.NET

O ASP.NET é a plataforma da Microsoft para o desenvolvimento de aplicações web e é o sucessor da tecnologia conhecida como ASP (*Active Server Pages*). É um componente do IIS (*Internet Information Services*) que permite, através de uma linguagem de programação integrada na plataforma .NET[17], criar páginas da aplicações web e serviços da *web* dinâmicos. O ASP.NET herda todas as características da plataforma .NET e as suas aplicações podem ser escritas em qualquer linguagem suportada pela mesma, sendo as mais comuns o Visual Basic e o C#.

- AJAX – Asynchronous JavaScript And XML

AJAX é o uso metodológico de tecnologias como o JavaScript e o XML, para tornar as páginas web mais interativas e mais rápidas, usando solicitações assíncronas de informações. O AJAX não é uma tecnologia, mas um conjunto de tecnologias conhecidas que trabalham em conjunto para criar novas funcionalidades no lado do cliente da aplicação.

- Microsoft SQL Server

O SQL server é um Sistema de Gestão de Bases de Dados (SGBD) desenvolvido pela Microsoft cuja principal função é armazenar e enviar dados conforme pedido por outras aplicações, quer estejam na mesma máquina ou em rede. As principais linguagens do Microsoft SQL são o Transact-SQL e o ANSI-SQL. Neste projeto é utilizada a primeira linguagem e todas as tabelas e modificações dos dados em ambiente de programação são feitos através do Microsoft Visual Studio.

Linguagens

Ao longo do projeto foram utilizadas diferentes linguagens, que se descrevem em seguida.

- C#

É uma linguagem orientada a objetos desenvolvida pela Microsoft para integrar a plataforma .NET é a linguagem principal do projeto. Como todas as linguagens orientadas a objetos, o C# oferece suporte aos quatro conceitos base das mesmas[13]:

Abstração	Capacidade de descrever cenários complexos através de conceitos simples, neste caso, através de classes.
Encapsulamento	Capacidade de ocultação dos detalhes de implementação de alguns métodos, sendo estes acedidos através de um interface[4].
Herança	Capacidade de criação de subclasses, permitindo a partilha de atributos e métodos de uma dada classe mais genérica[13].
Polimorfismo	Capacidade de utilizar o mesmo nome para um método em diferentes classes.

Tabela 6: Caraterísticas das linguagens orientadas a objetos

- HTML - *Hyper Text Markup Language*

É uma linguagem de marcação utilizada para produzir páginas web e que podem ser interpretados por navegadores de internet. A sua estrutura assenta em marcas entre parênteses angulares ('<','>') e esses são os comandos de formatação da linguagem.

- CSS – *Cascade Style Sheet*

O CSS, *Cascade Style Sheets*, é uma linguagem de folhas de estilo usado para descrever a semântica de apresentação de um documento escrito, por exemplo, em HTML. A sua principal aplicação é o desenvolvimento de páginas web, mas pode ser aplicada a qualquer tipo de documento XML. Foi criada para permitir uma separação entre o conteúdo do documento da sua apresentação, incluindo elementos como o esquema de página, cores e alinhamentos. Assim melhora-se o acesso ao conteúdo e a flexibilidade e controlo das especificações das características da aplicação. Também permite a reutilização da mesma formatação por várias páginas diferentes e a adaptação de elementos como o tamanho da aplicação dependendo do navegador utilizado[17].

- JavaScript

Javascript (JS) é uma linguagem de *script* da família de linguagens C e é atualmente a principal linguagem para programação do lado do cliente em navegadores web e que permite a interatividade[17] na página. É uma linguagem orientada a objetos, que oferece uma tipagem dinâmica, ou seja, os tipos de variáveis não precisam estar definidos; é interpretada, ao invés de ser compilada e oferece um bom suporte a expressões regulares. O Javascript também permite modificar os estilos dos elementos HTML da página e atributos das folhas de estilo[10].

Bibliotecas externas ao .NET

Para desenhar as cablagens tornou-se necessário utilizar uma biblioteca externa que facilitasse a representação de grafos, pelo que se fez uma pesquisa acerca das possibilidades existentes e que pode ser consultada no anexo B. A biblioteca utilizada foi a biblioteca WireIt em JavaScript.

A biblioteca WireIt é uma biblioteca em Javascript *open-source* que tem como objetivo permitir o desenho de fios para aplicações visuais, editores de grafos, modelação gráfica, etc[20].

Foi a biblioteca escolhida para este projeto, por ser leve, simples e por utilizar uma linguagem diferente das descritas no Anexo B, que estão em C#. Um fator decisivo foi também o facto de não estar cingido a um conjunto de estruturas de dados, uma vez que opera no lado do cliente. Para este projeto apenas foram usadas as funções de desenho simples da biblioteca, que funciona através do conceito de fios e terminais (ver figura 32), podendo estes ser vistos como arestas e vértices de um grafo não orientado. Adaptou-se o código original às necessidades do projeto pelo que foi adicionada a numeração de terminais (vértices) e foram modificados outros pormenores.

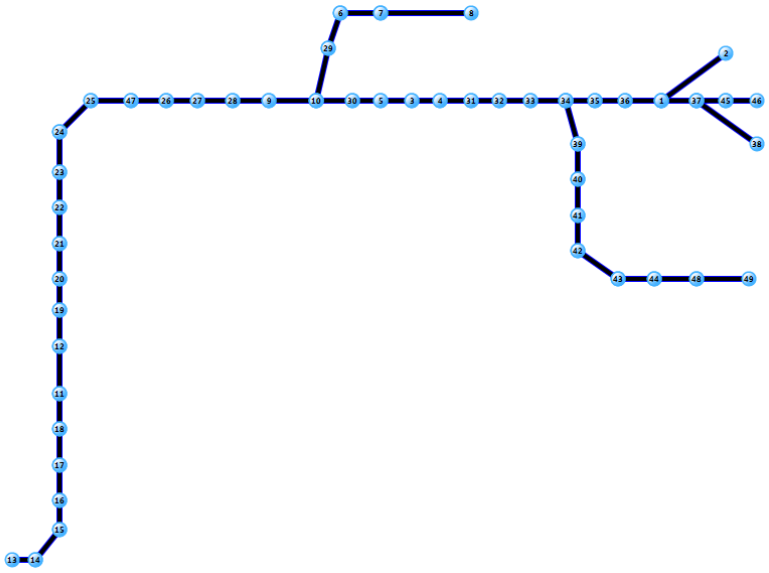


Figura 32: Exemplo de desenho feito com a biblioteca *WireIt* adaptado

Capítulo 3 CheckPGM - A solução

Neste capítulo são descritas em maior pormenor algumas particularidades da implementação da solução, como é o caso da criação da camada de negócio, a identificação da estrutura do ficheiro DSI com expressões regulares e o tratamento de erros, finalizando com uma descrição do interface com o utilizador. Optou-se por uma arquitetura em três camadas (ver secção 2.2), que é descrita em seguida.

3.1. Camada de negócio

Foi desenvolvida uma componente intermédia que serve de ligação entre o interface de utilizador e a base de dados, aumentando assim a segurança dos mesmos e a organização do código. Contém todas as regras de negócio e a camada de acesso a dados, que, por sua vez, contém todos os métodos com instruções SQL. A figura seguinte mostra um esquema que representa a organização da camada de negócio, onde existem três componentes:

1. Conjunto de classes intermédias (nº1 da figura 33), cada uma delas referidas como *General Class* (ver figura 34), que recebem os pedidos através de um interface e chamam um método da classe *UtilDB* (nº3 da figura 33, figura 34) para que esta devolva os dados pedidos;

2. Classe que contém os métodos de alocação de objetos (chamada *Factory*) de cada uma das classes intermédias (nº 2 da figura 33);

3. Classe que contém todos os métodos de acesso a dados (*insert, update, delete, select*, etc) chamada *UtilDB*. A maioria dos métodos contidos nesta classe são privados de maneira a reforçar a segurança dos mesmos.

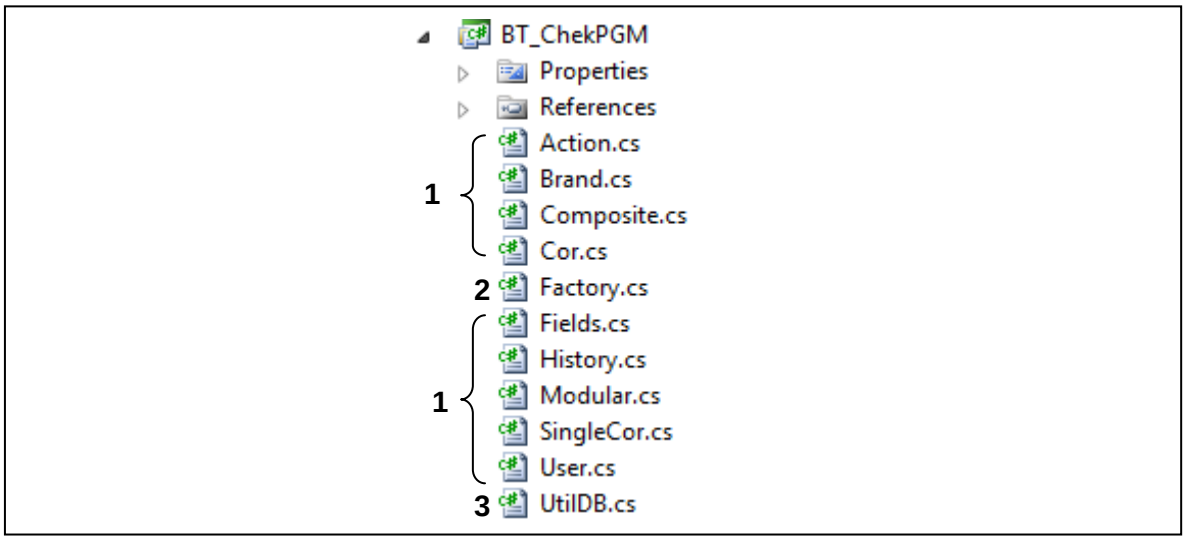


Figura 33: Esquema da camada de negócio

Estas componentes vão comunicar entre si para formar a camada de negócios. Cada classe (da gama *General Class,* figura 34) tem um método de instanciação associado na classe *Factory* e vai comunicar com a base de dados através dos métodos contidos na classe *UtilDB*, como se observa no esquema seguinte:

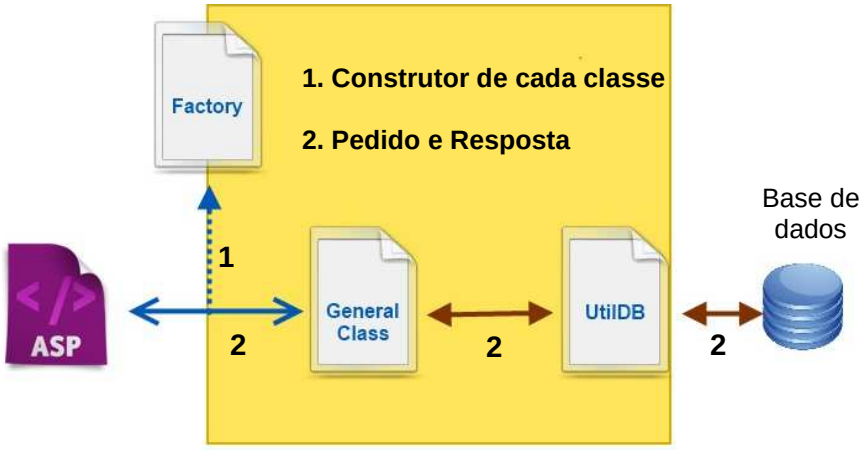


Figura 34: Esquema de comunicação entre partes da camada de negócio

Como o esquema mostra, as páginas ASP não têm acesso direto aos dados, nem os conseguem manipular sem passar pelos métodos intermédios, o que previne a exposição da estrutura da base de dados e o uso de técnicas de *hacking* como *SQL Injection* (técnica que envolve a introdução de comandos SQL em caixas de input de texto).

Na figura 35 vemos a aplicação de dois conceitos-chave da programação orientada a objetos: abstração, na criação de classes genéricas; encapsulamento, na definição de interfaces para cada uma das classes (caso da classe *Action*, figura 35).

```
public interface IAction
{
    DataSet getAction(int id);
}

internal class Action : IAction
{

    string nameTable = "Actions";
    public DataSet getAction(int id)
    {
        string[] names = new string[1] { "Action" };
        string[] cond = new string[1] { "ID_Action = '" + id + "'" };
        return UtilDB.getWithCond(names, cond, nameTable); }

}
```

Figura 35: Exemplo de uma classe genérica - *Action*

3.2. Identificação da estrutura do ficheiro DSI

A informação a extrair do ficheiro provém de uma secção única, identificada através do símbolo '%' das suas linhas subsequentes. Para realizar o processamento do ficheiro e detetar este tipo de estrutura recorreu-se ao uso de expressões regulares através da biblioteca *Regex* já incluída na linguagem C#. Com um conjunto de operadores Regex, são definidos padrões, ou seja, são criadas expressões reconhecedoras de uma determinada estrutura. Para este projeto existem três padrões principais que reconhecem

secções, linhas de secções e secções vazias. Pretende-se portanto detetar uma dada secção e posteriormente tratar a informação nela contida com funções auxiliares. Foi criada uma classe chamada "__File" que contém todos os métodos de deteção e de filtração de secções, uma vez que os métodos são utilizados em várias páginas web do projeto.

- Expressões regulares

Uma dada linguagem é regular se puder ser definida por uma expressão regular[1] e portanto, se for reconhecida por um autómato finito. Representar uma linguagem regular através do seu autómato correspondente nem sempre é uma tarefa fácil quando o número de estados é elevado[1]. São expressões com símbolos pertencentes a um dado alfabeto e símbolos operadores, carateres esses que estão representados na tabela seguinte, resumidamente[8]:

Símbolo	Função
.	Reconhece qualquer carater
[ ]	Reconhece a ocorrência de um caracter contido na lista.
[^]	Reconhece qualquer carater que não esteja contido na lista
^	Precede o carater que deve ser encontrado no início do texto a analisar
$	Sucede ao carater que deve ser encontrado no fim do texto a analisar
-	Especifica um intervalo de carateres, por exemplo: [a-z0-9]
+	Repete o grupo ou carater anterior uma ou mais vezes
*	Repete o grupo ou carater anterior 0 ou mais vezes
?	O carater ou grupo anterior é opcional
{n,m}	Repete o carater ou grupo anterior no mínimo n vezes e no máximo m vezes

Tabela 7: Caracteres da sintaxe das expressões regulares usados

No ambiente de programação .NET, e no caso da linguagem C#, existem funções que, dado um padrão formado por estes carateres, pesquisa num bloco de texto o conjunto de cadeias de carateres que conferem o padrão. A nível do projeto, este método está presente em vários pontos e para realizar diferentes tarefas (comparação de *strings*, verificação de *inputs*, entre outros), no entanto a sua principal função é o reconhecimento de secções e informações pertinentes do ficheiro DSI. Para comparação destes padrões, faz-se a leitura na íntegra do ficheiro para uma variável do tipo *string* e procede-se à comparação com cada um dos diferentes padrões. Para uma melhor visualização das expressões utilizadas seguem-se os diagramas de transição associados às expressões usadas, nas figuras 36, 37 e 38.

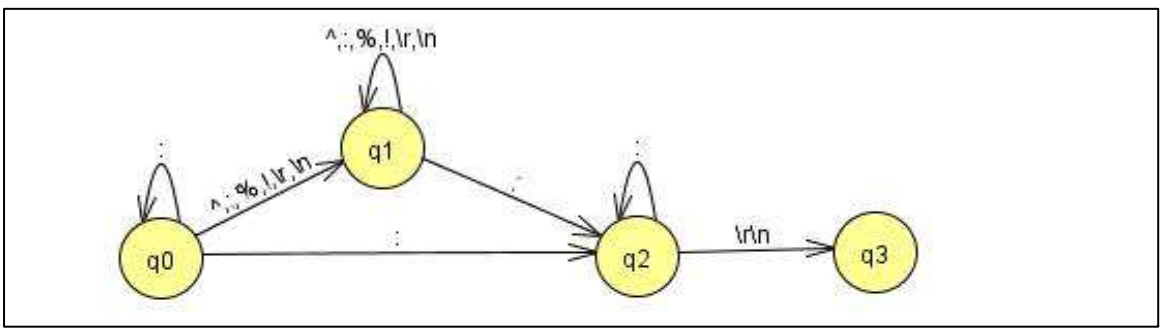


Figura 36: Autómato reconhecedor de uma linha de secção

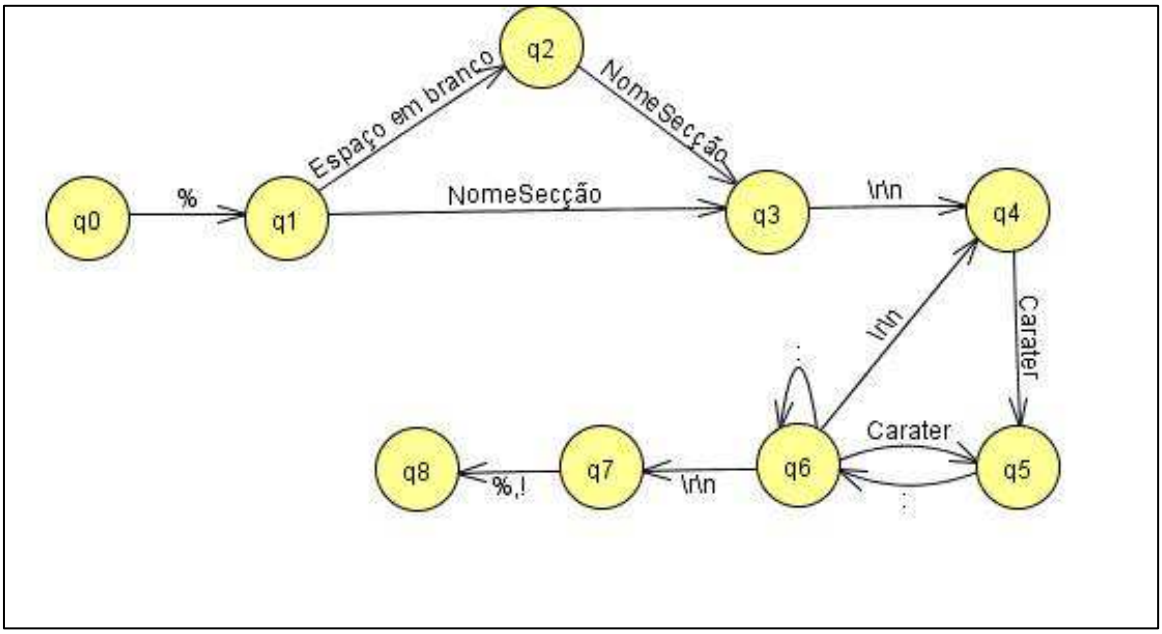


Figura 37: Autómato reconhecedor de uma secção inteira

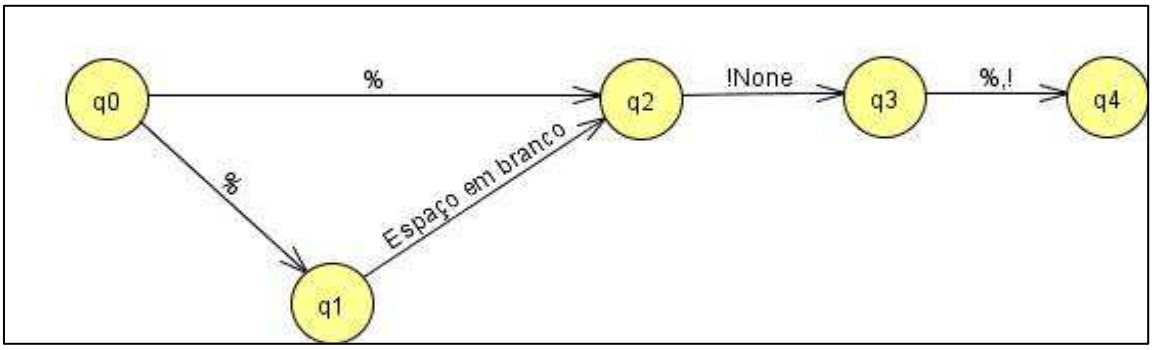


Figura 38: Autómato reconhecedor de uma secção vazia

3.3. Desenho de cablagens

Para complementar a aplicação decidiu-se implementar também uma funcionalidade que representasse graficamente a cablagem contida do ficheiro DSI, cuja informação se encontra em secção própria que engloba as coordenadas cartesianas de cada ponto (componente) da cablagem. Para o desenho de cablagens recorreu-se a uma biblioteca externa em Javascript chamada *WireIt*[20]. Para ser utilizada foi necessário trabalhar a informação referente aos pontos da cablagem e transformá-la no *input* a ser enviado para as funções do lado do servidor. Para isso criaram-se classes novas e algoritmos auxiliares que são descritas seguidamente.

Classes Auxiliares

Dado que a informação sobre os pontos das cablagens consiste em coordenadas cartesianas (x e y) de cada extremo de cada ramo, adotou-se a representação através de um grafo não orientado. Para auxiliar a construção do grafo através de coordenadas de

pontos extremos das arestas criaram-se duas classes para representar pontos e linhas denominadas *Ponto* e *Linha*, respetivamente.

- Ponto

Tal como o nome indica, esta classe tem por objetivo representar um ponto através das suas coordenadas cartesianas x e y, como se observa na tabela 8. Para efeitos de identificação dos mesmos, criaram-se outros atributos complementares para cada instância desta classe, sendo eles: *nome* e *id*. O atributo 'nome' contém o nome dado ao nó na cablagem e o atributo 'id' indica qual o número identificativo do mesmo, número este resultante de uma classificação posterior, não contida no ficheiro DSI, mas que facilita a visualização da cablagem aquando do seu desenho.

Ponto
x
y
nome
id

Tabela 8: Tabela da classe Ponto

- Linha

Classe que representa uma linha ou ramo da cablagem. É constituída pelos pontos extremos do mesmo, bem como por atributos com informação complementar, não contidos no ficheiro original, como se observa na tabela seguinte:

Linha
Ponto inicial
Ponto final
Nome
Cor
Secção

Tabela 9: Tabela da classe Linha

Algoritmos de cálculo de caminhos

Mais tarde, surgiu a necessidade de determinar qual seria o percurso entre dois pontos da mesma cablagem (caso *composite*), sabendo que existe apenas um percurso possível e apenas tendo informação sobre cada aresta da cablagem completa, isto é, ponto inicial e ponto final. Criou-se então um algoritmo baseado num algoritmo de busca em profundidade.

Dado que nenhum ramo da cablagem forma um circuito fechado, considera-se que o grafo a desenhar é uma árvore. Seja G uma árvore com $V=\{v_1,v_2,...,v_n\}$ o conjunto de vértices e $E=\{e_1,e_2,...,e_{n-1}\}$ o conjunto das arestas. Pretende-se encontrar o caminho entre dois vértices v_i e v_f .

- Descrição

  Cada vértice v é visitado e calculam-se os seus vizinhos. Então, cada vértice adjacente a v que não seja o seu antecessor é visitado. Se um vértice v não tem vértices adjacentes (é terminal) ou se o único vértice adjacente é o antecessor, o algoritmo volta para trás (*backtracking*), para o antecessor de v. O percurso está terminado se esse processo de visita e retorno chega ao vértice final.

- Pseudo-código

  Para a implementação do algoritmo é necessária uma função auxiliar, cujo algoritmo se pode analisar na figura seguinte, que retorne os vizinhos de um dado vértice, chamada *GetVizinhos* onde, dados o ponto pretendido v, o seu antecessor, todos os ramos possíveis e o número de ramos, devolve uma lista de pontos com os vértices vizinhos de v fazendo um percurso exaustivo do *array* de ramos.

```
GetVizinhos(Ponto antecessor, Ponto v, Linha[] ramos, int numRamos)
{
        i = 0;
        vizinhos = Ponto[];
        Para todos os i < numRamos
        {
                if ( v == ponto inicial do ramo i)
                { //Encontrou o ponto e portanto um vizinho
                        If( ponto final do ramo i != antecessor)
                                Adicionar ponto final a  vizinhos;
                }
                if (v == ponto final do ramo i)
                {//Encontrou o ponto e portanto um vizinho
                        if ( ponto inicial do ramo i != antecessor)
                                Adicionar ponto inicial a  vizinhos;
                }
        }
        if (não encontrou vizinhos)
                Return null;
        Return vizinhos;
}
```

Figura 39: Algoritmo da função *GetVizinhos*

A função *GetCaminho* (função principal) tem como objetivo retornar o percurso entre dois vértices sob a forma de uma *string*. Dado que se lida com uma árvore, esse caminho é único. Como se pode ver na figura que se segue, os parâmetros de entrada desta função são: o ponto antecessor, o ponto inicial, o ponto final, o *array* de todos os ramos (ou arestas) do grafo e o número de ramos total do grafo. É de notar que o ponto inicial é o ponto a ser testado, o ponto antecessor é o ponto, imediatamente antecessor a este, através do qual se chegou ao ponto inicial (e portanto na primeira chamada da função vai ser *null*) e o ponto final é o ponto de destino. Esta função é recursiva e o seu critério de paragem é atingido quando o ponto inicial coincide com o ponto final: nesse caso devolve uma *string* com as coordenadas do ponto antecessor juntamente com as coordenadas do ponto final, separadas por uma vírgula. Caso o ponto inicial e final não coincidam, vai ser testada a existência de vizinhos para o ponto 'inicial', descartando o ponto

antecessor como vizinho e, ou não existem vizinhos, e é retornado o valor *null*, ou então é guardada na variável 'viz' um *array* com todos os vizinhos do ponto 'inicial'.

Para cada um desses vizinhos vai ser chamada a função *GetCaminho*, mas como estamos a avançar na pesquisa, o ponto antecessor passa a ser o ponto 'inicial' e o ponto a ser testado (o ponto 'inicial') passa a ser o vizinho atual. Se pelo caminho obtido através do ramo do vizinho atual (vizinho "i"), não for encontrado nenhum caminho que leve ao ponto final, então vai ser descartado esse caminho e passa-se ao vizinho seguinte. Caso o caminho leve até um caso em que o ponto inicial e final coincidam, vai ser construída a *string* do caminho, chamada *Road*, concatenando todas as coordenadas dos pontos desse caminho.

```
GetCaminho(Ponto antecessor, Ponto inicial, Ponto final, Linha[] ramos, int numRamos)
{
        //Booleano que indica que se encontrou o ponto final
        Boolean found = false;
        //String com o caminho
        string road = null;

        //Caso ponto inicial= ponto final: critério de paragem
        if (isEqual(init, fin))
        {
            return fin.x + ";" + fin.y + ";" + fin.nome;
        }
        else // Ou é terminal(não tem vizinhos) ou tem vizinhos
        {
            //Determinar os vizinhos
            Ponto[] vizinhos = GetVizinhos(antecessor, init, ramos, tamanho);
            if (vizinhos == null) // É terminal
                return null;
            else //Tem vizinhos
            {
                int i = 0;
                found = false;
                //Enquanto não encontrar um caminho ou tiver vizinhos
                while (!found && i < vizinhos.Length)
                {
                    //Buscar o caminho a partir de cada um dos vizinhos do ponto inicial
                    string cam = GetCaminho(init, vizinhos[i], fin, ramos, tamanho);
                    //Se retornar nulo, não existe caminho e passa-se ao vizinho seguinte
                    if (cam == null)
                        i++;
                    else
                    { // Caso contrário concatena-se o ponto inicial ao caminho já existente
                        found = true;
                        return road = init.x + ";" + init.y + ";" + init.nome + ";" + cam;
                    }
                }
            }
        }
        //Caso não encontre caminho ou esgote os vizinhos retorna nulo
        if (!found) road = null;
        return road;
}
```

Figura 40: Algoritmo da função *GetCaminho*

O algoritmo anterior cobre a maioria das cablagens, no entanto existe a possibilidade, apesar de ser rara, de a cablagem conter circuitos fechados e portanto, o grafo correspondente conter ciclos. Para tratar esse caso foi necessário implementar um segundo algoritmo que calcula o caminho mais curto entre dois pontos v_i e v_f. Implementou-se uma versão do conhecido algoritmo de Dijkstra, que será descrito em seguida. Este algoritmo foi concebido pelo cientista da computação holandês Edsger Dijkstra em 1956 e publicado em 1959.

- Descrição

  Este algoritmo funciona através do uso de três estruturas de dados:

  1. *Array* 'dist' que contém as distâncias percorridas para cada vértice.
  2. *Array* 'path' que contém o vértice antecessor de cada vértice para se poder construir o caminho final.
  3. Lista com os vértices que não foram ainda visitados.

  Como se indicou na tabela 8, cada ponto tem associado um número de identificação único, pelo índice das variáveis 'dist' e 'path' correspondem a esses números em todo o algoritmo.

- Pseudo-código

  Tal como o algoritmo anterior, também este necessita de uma função auxiliar. O objetivo dela é determinar qual será o próximo vértice escolhido, tendo em conta o comprimento do caminho já percorrido, que deverá ser mínimo. A função *GetNextVertex* tem como parâmetros de entrada uma lista L, que é a lista dos vértices que ainda não foram visitados, e o *array* das distâncias atuais dos vértices é o 'dist'. A função retorna o índice do *array* para o qual o vértice correspondente tem o valor de distância mínimo, como se observa na figura seguinte:

```
GetNextVertex (List<int> L,double[] dist)
{
        double min =   ∞   ;
        int vertex = -1;
        Para cada vértice j na lista L
        {
            //procurar a menor distância existente
            if (dist[j] <= min)
            {
                min = dist[j];
                vertex = j;
            }
        }
      return vertex;
}
```

Figura 41: Algoritmo da função *GetNextVertex*

A função *Dijkstra* (função principal) tem como parâmetros de entrada um *array* bidimensional, que corresponde à matriz de adjacência que contém os pesos entre os vértices (distâncias euclidianas entre os pontos), e um inteiro, que é o

índice do vértice inicial. É feita a inicialização de todos os vértices com as distâncias como "infinito" (exceto o vértice inicial, cuja distância é colocada a zero) e são adicionados à lista L, uma vez que não foram ainda visitados. De acordo com o algoritmo da figura seguinte, é escolhido o primeiro vértice através da função *GetNextVertex* e para cada vértice adjacente ao mesmo é determinado qual ficará com uma distância mínima a partir do vértice atual[22]. Esse vértice cuja distância calculada é mínima, será adicionado ao *array* 'path' e o processo de escolha de vértices através da função auxiliar é repetido até que não haja mais vértices por visitar[22].

```
Dijkstra(double[,] adj, int init)
{
 //Lista que contém todos os nós que ainda não foram visitados
   List<int> L = new List<int>();

     //Inicialização de valores

     Para todo o v pertencente a V[G]
     {
                    dist[v]= ∞ ;
                    L.add(v);
     }
     dist[init]=0;
     path[init]=-1;


     enquando L!=vazio
     {
                    u = GetNextVertex(L,dist);
                    L = L\{u};
                    para cada v adjacente a u
                    {
                          if (dist[v] > dist[u] + adj[u,v])
                          {
                            dist[v] = dist[u] + adj[u,v];
                            path[v] = u;
                          }
                    }
     }
}
```

Figura 42: Algoritmo da função Dijkstra

3.4. Permissões

A identificação de cada utilizador é feita através de uma componente interna da empresa que acede ao servidor de utilizadores e retorna o nome de utilizador da pessoa que está a utilizar máquina em questão.
Existem apenas dois tipos de utilizadores da aplicação: administrador e utilizador. O administrador é aquele que tem mais privilégios, na medida em que terá acesso a páginas exclusivas e a manipulação de alguns dados que não é permitido aos outros utilizadores. Para implementar estas restrições criou-se uma tabela chamada *Users* (ver modelo do domínio, figura 28) que contém o nome de utilizador interno da empresa de cada pessoa irá usar a aplicação e lhe associa um nível de permissões, como se verifica na tabela 10.

Users
UserName
Level

Tabela 10: Tabela dos utilizadores

Níveis de permissão

- Nível 1: nível mais baixo de permissões, corresponde ao utilizador;
- Nível 2: nível mais alto de permissões, concede acesso de administrador.

O nível de permissão é verificado em todas as páginas relevantes e caso não seja concedido o acesso, a aplicação redirecionará para uma página de erro, notificando do sucedido através de uma mensagem de texto.

3.4. Tratamento de erros

O tratamento dos erros de uma aplicação é vital para sua segurança. Umas das formas mais comuns de se burlar um sistema, qualquer que seja, é forçando um erro inesperado na sua execução. Quando um erro inesperado acontece, se a aplicação não estiver devidamente "preparada", pode acontecer o escape de informações que revelam a arquitetura do sistema através das mensagens de erro na linguagem onde o sistema foi desenvolvido.

Na aplicação foi feito o tratamento de erros através de práticas como:o uso das instruções *try, catch* e *finally*, a programação defensiva e páginas de erro que serão descritas em seguida.

3.4.1. *Try..catch..finally*

O tratamento de erros baseia-se principalmente no tratamento de falhas de sistema. A falha de sistema mais geral é chamada *Exception*. Estas falhas podem causar perdas de informação e interrupções de processos das quais não se pode recuperar[6]. Para tratar blocos de código com possibilidades de erros, usamos o conjunto *try, catch e finally* (ver figura 43) que podem afetar diretamente o comportamento do software durante e após a ocorrência de destes. A instrução *try..catch* permite ao programador gerir as execuções do código, obrigando a que, na ocorrência de uma exceção, esta possa ser tratada em tempo de execução. Tem a seguinte estrutura:

```
try
{       //Código passível de erro }
catch(Exception e)
{       // Caso o código do try dispare uma exceção, este código será executado   }
finally
{     //Este código é sempre executado, quer tenham ocorridos erros ou não.    }
```

Figura 43: Sintaxe das instruções *try,catch e finally*

A nível do projeto, este tipo de tratamento de erros é predominante nos métodos com instruções de SQL (*insert, update* e *delete*) que são aqueles que mais exceções causam.

3.4.2. Programação defensiva

A programação defensiva é um padrão de desenvolvimento que tem em conta uma antecipação dos possíveis erros que possam surgir, antes que estes aconteçam, como por exemplo, o aparecimento de valores nulos. Assim aumenta-se a segurança e também a qualidade da aplicação. Esta medida foi tomada no desenvolvimento do projeto, sempre que possível. Na figura seguinte está um exemplo de validação de dados de entrada, para impedir valores incorretos, evidenciando a programação defensiva.

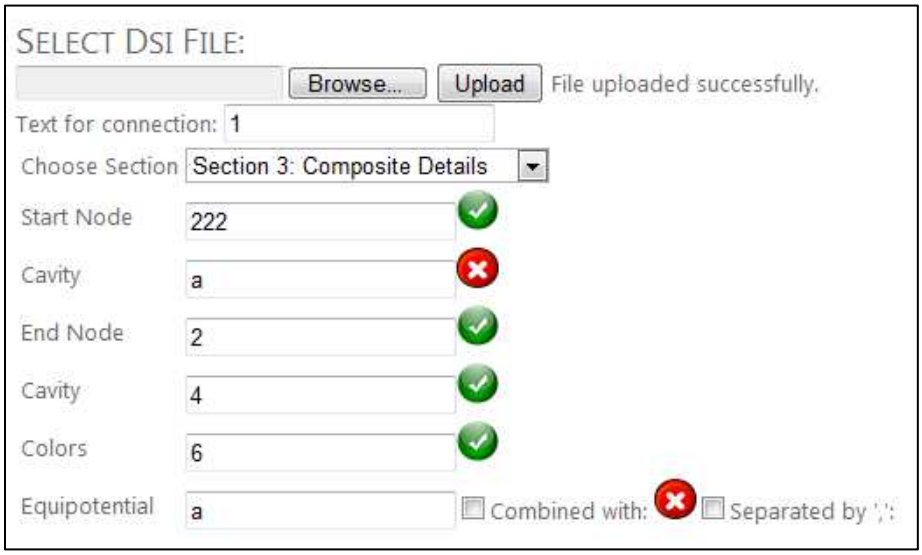


Figura 44: Exemplo de programação defensiva: correção de inputs

Outro exemplo deste tipo está presente em várias páginas da aplicação onde é necessário carregar ficheiros DSI (na geração de ficheiros EDT) ou EDT (editor de ficheiros EDT). Caso o utilizador carregue um ficheiro cuja extensão não seja a pretendida, aparece uma mensagem que indica que ocorreu um erro, que é descrita em seguida.

3.4.3. Página de erro

Quando acontece uma exceção normalmente a aplicação mostra um ecrã amarelo com a descrição do erro sucedido. Para evitar que o usuário veja esta página, cuja informação não lhe é útil, redireciona-se a aplicação para uma página de erro que mostra uma mensagem que alerta o utilizador da ocorrência de um erro. Para fazer com que haja redirecionamento é necessário adicionar o seguinte código ao ficheiro *web.config* da aplicação:

```
<customErrors mode="On" defaultRedirect="ErrorPage.aspx" />
```

Figura 45: Linha de código de ativação da página de erro

Assim, cada vez que ocorre um erro de qualquer natureza, aparecerá a uma página com uma mensagem que indica o sucedido.

3.5. Interface

Pretende-se que o interface da aplicação tenha um aspeto profissional, com cores apelativas e funcionais. A aplicação também terá de ser intuitiva e organizada, bem como ter uma estrutura consistente ao longo das páginas.

3.5.1. Aspeto geral (*Layout)*

À exceção da página inicial e da página de erros, todas as outras páginas seguem o mesmo layout. O *layout* é definido através de uma página principal (*master page*), em conjunto com uma folha de estilo (CSS) que definem todos os elementos estéticos e esquemáticos comuns às páginas em questão. A aplicação foi otimizada para o Internet Explorer 8, pelo que alguns elementos podem aparecer diferentes consoante o *browser* onde se acede à aplicação. Como a imagem 46 mostra, o *layout* é constituído por 4 elementos:

1. Cabeçalho: contém o logotipo da aplicação (esquerda) e o nome do utilizador da máquina onde a aplicação está a ser acedida;

2. Menu: contém o acesso a todas as páginas da aplicação, sendo que quando se acede a sub-menus ou às páginas em si, a aba correspondente fica assinalada a cinza escuro;

3. Conteúdo: como o nome indica aqui é colocado todo o conteúdo necessário a cada página em questão;

4. Rodapé: no rodapé apenas consta alguma informação útil ao utilizador: hora, dia da semana e data atual.

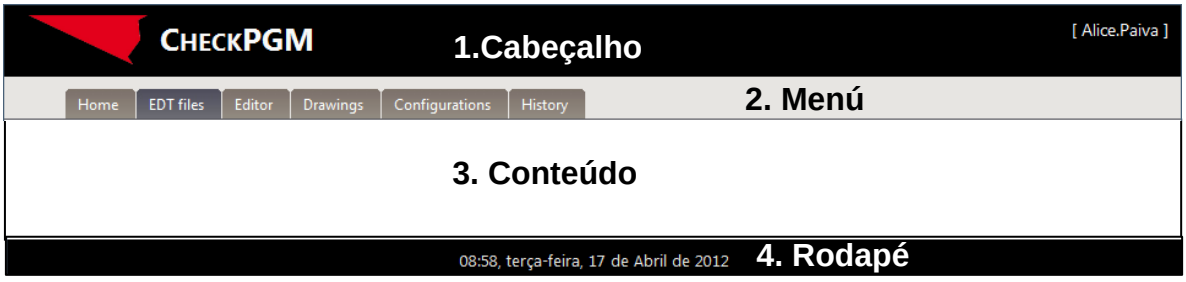


Figura 46: Esquema de página

A página inicial em conjunto com a página de erros têm uma *master page* própria, que em tudo é similar à referida acima com exceção do menu, que não está presente, como se verifica na imagem seguinte:

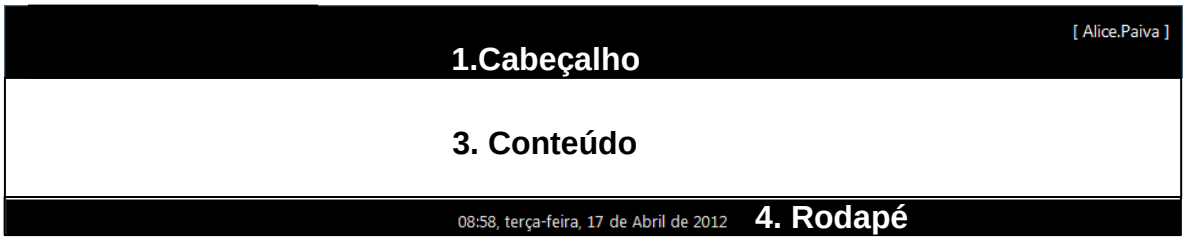


Figura 47: Esquema da página inicial e de erro

3.5.2. Mapa da aplicação

A aplicação contém um conjuntos de páginas que podem ser acedidas diretamente através do menu inicial ou através de sub-menus. A imagem seguinte mostra a relação existente entre as diferentes páginas e a organização da aplicação.

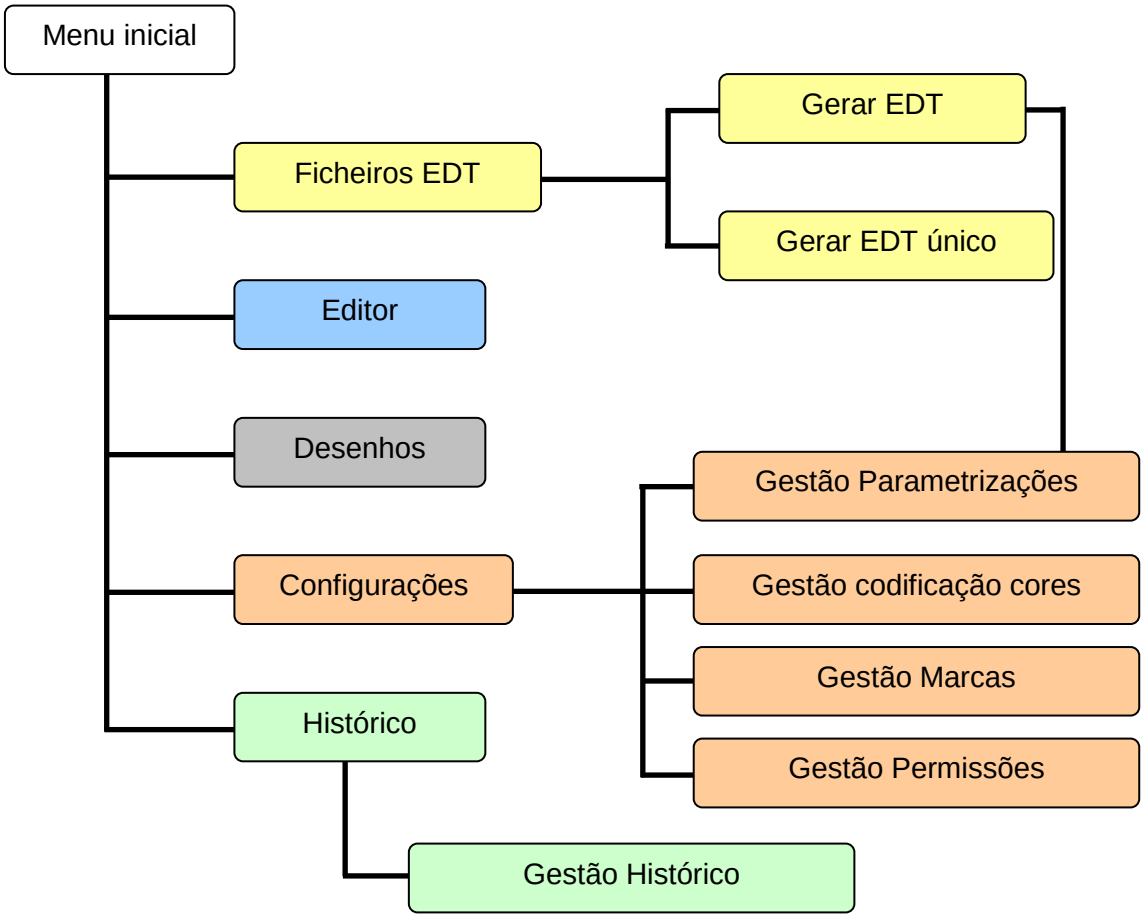


Figura 48: Mapa da aplicação

Menu inicial

O menu inicial não segue o esquema das restantes páginas da aplicação, pelo que tem uma *master page* e uma folha de estilos única para definir o seu aspeto. A página inicial tem no seu conteúdo uma série de hiperligações, através de ícones, para cada um dos sub-menus disponíveis, como mostra a figura seguinte.

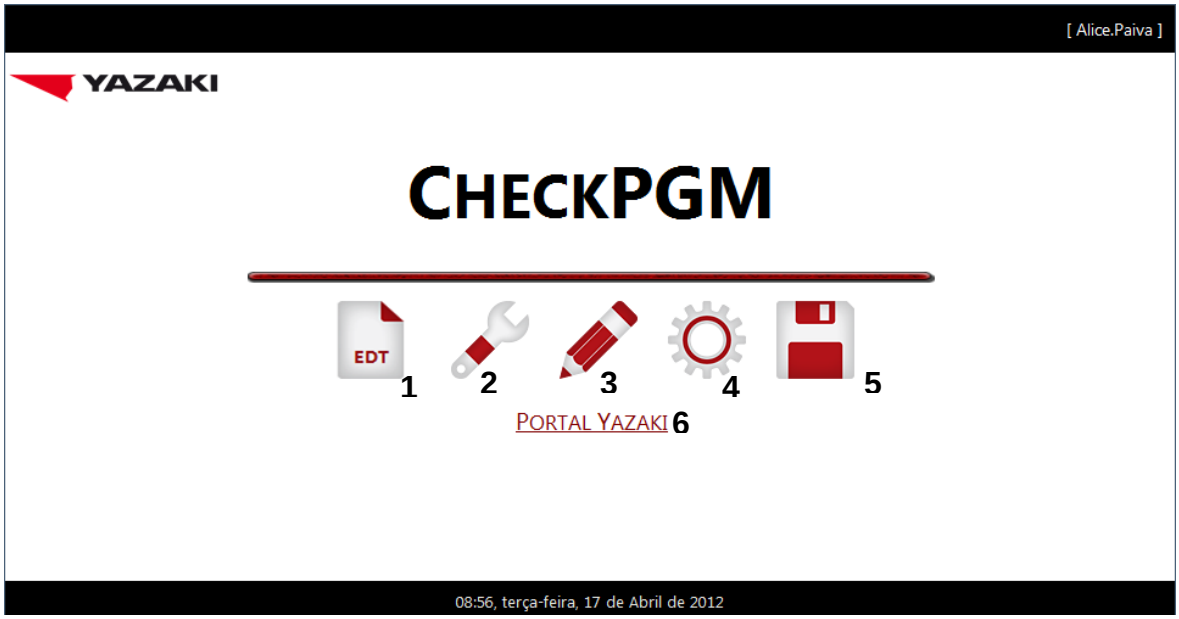


Figura 49: Menu inicial da aplicação

Legenda da figura 49:

1. Hiperligação para o menu das operações sobre ficheiros DSI;
2. Hiperligação para o editor de ficheiros DSI;
3. Hiperligação para o menu dos desenhos de cablagens;
4. Hiperligação para o menu das configurações;
5. Hiperligação para o menu do histórico;
6. Hiperligação para o portal interno da empresa (externo à aplicação);

Menu Ficheiros EDT

O primeiro sub-menu é aquele que contém as funcionalidades referentes aos ficheiros EDT e tem a aparência seguinte:

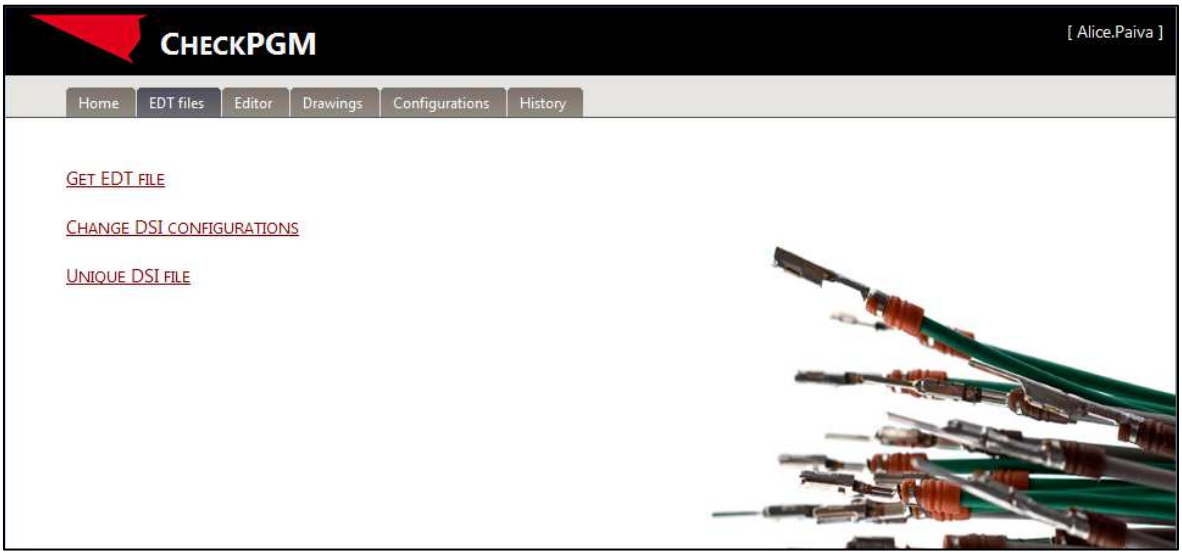


Figura 50: Menu Ficheiros EDT

A partir desta página pode-se aceder às páginas de geração de ficheiros EDT a partir das parametrizações e sem elas (figuras 51 e 52, respetivamente), bem como aceder às configurações das parametrizações (figura seguinte).

- Gerar ficheiro EDT a partir de parametrizações

  Aqui é necessário escolher uma marca e um modelo para saber quais as parametrizações e codificação de cores a usar. Pode-se então gerar o ficheiro aberto e guardá-lo em local a escolher ou abri-lo diretamente no editor de ficheiros EDT. A imagem seguinte ilustra as diferentes funcionalidades presentes nesta página:

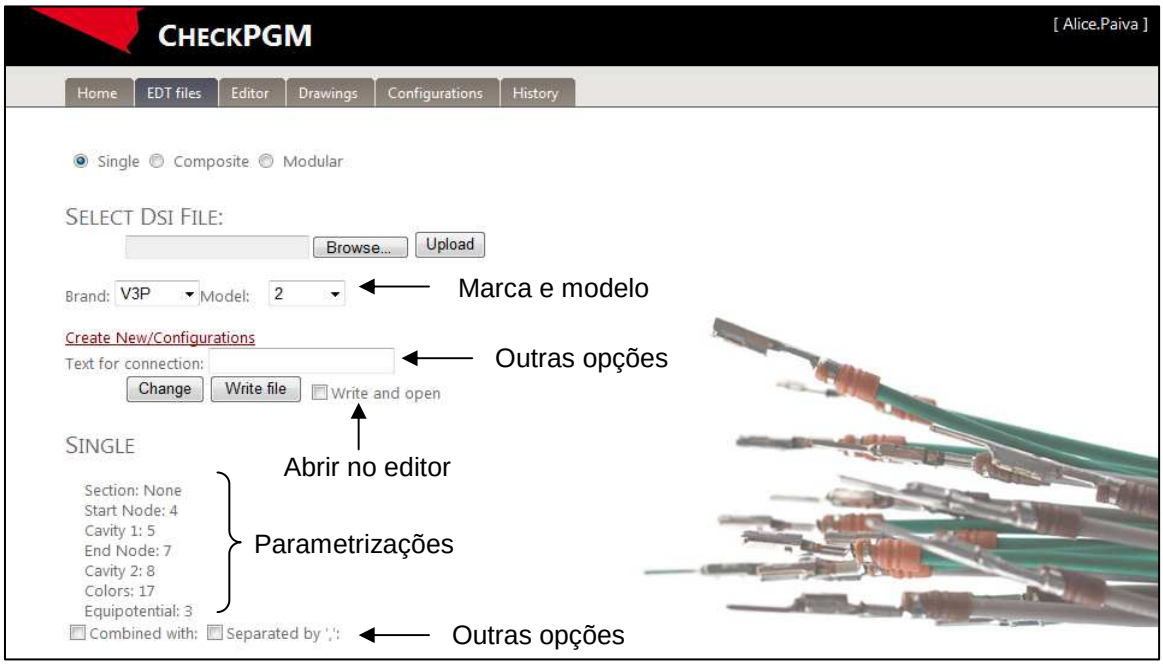


Figura 51: Página de geração de ficheiros EDT através de parametrização

- Gerar ficheiro EDT único

  Para gerar um ficheiro EDT sem usar as parametrizações guardadas na base dados criou-se uma página onde as mesmas pudessem ser introduzidas manualmente, para os casos especiais e mais raros do ficheiro. A base de funcionamento é a mesma da página anterior, como se observa na figura 52.

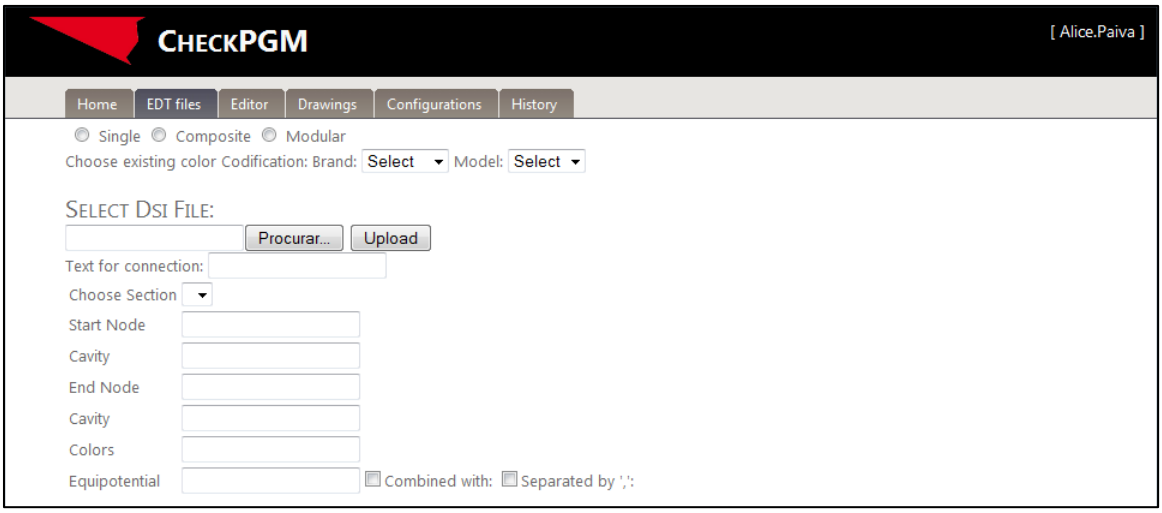
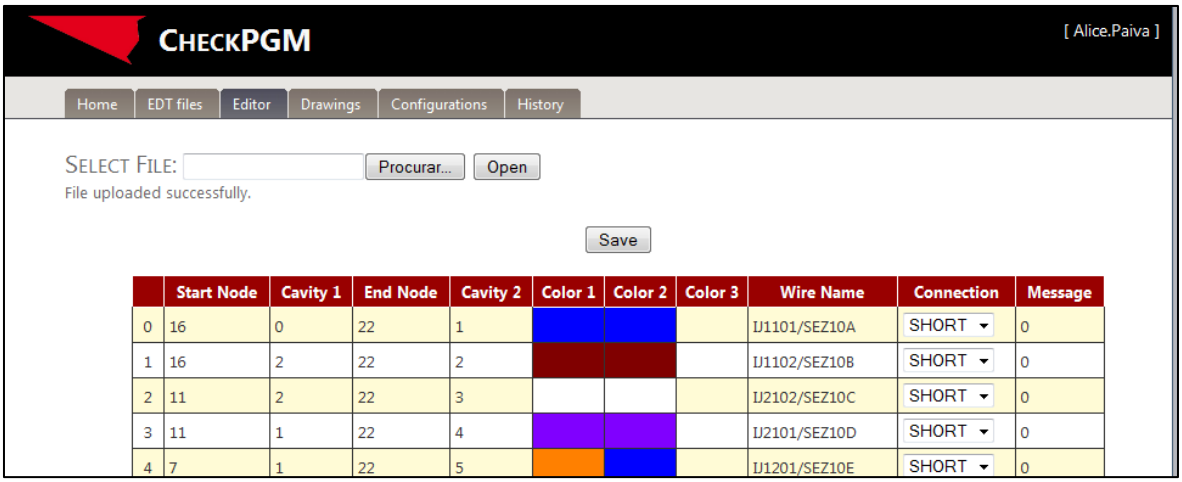


Figura 52: Página de geração de ficheiro EDT com parametrização única

Editor

O segundo menu contém o editor de ficheiros EDT, que disponibiliza a informação nele contida simulando uma folha de cálculo (figura 53), permitindo que sejam feitas alterações e salvar as mesmas num novo ficheiro EDT.



	Start Node	Cavity 1	End Node	Cavity 2	Color 1	Color 2	Color 3	Wire Name	Connection	Message
0	16	0	22	1				IJ1101/SEZ10A	SHORT	0
1	16	2	22	2				IJ1102/SEZ10B	SHORT	0
2	11	2	22	3				IJ2102/SEZ10C	SHORT	0
3	11	1	22	4				IJ2101/SEZ10D	SHORT	0
4	7	1	22	5				IJ1201/SEZ10E	SHORT	0

Figura 53: Editor de ficheiros EDT

Desenhos

Na página desenhos estão todas as ferramentas necessárias para a visualização das cablagens. O menu "base" tem a estrutura mostrada na figura seguinte:

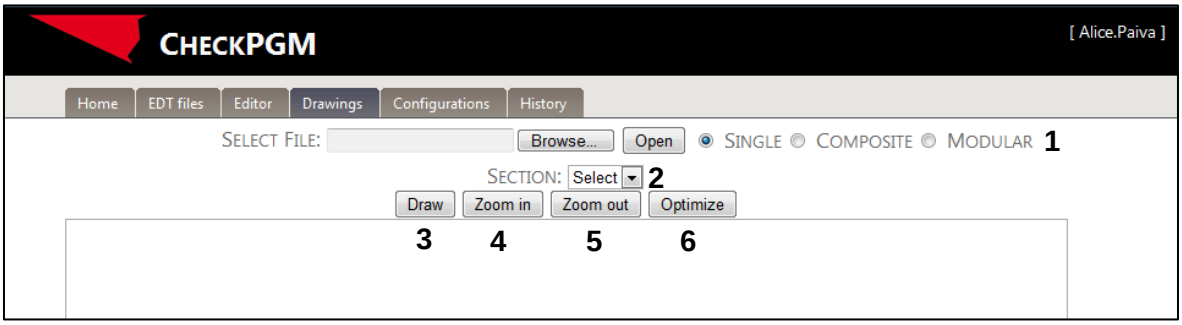


Figura 54: Página de desenho de cablagens

Onde:

1. Menu de tipo de ficheiro DSI;
2. Seleção de secção que contém informação dos pontos;
3. Botão para desenhar;
4. Botão para ampliar o desenho (1.5x);
5. Botão para reduzir o desenho (0.5x);
6. Botão para ajustar o desenho à área de desenho.

Consoante o tipo de ficheiro DSI selecionado na zona 1 da página, irão aparecer diferentes opções de desenho.

- Caso *Composite*

  Aqui, é necessário selecionar a marca e o modelo pretendidos, para detetar as parametrizações necessárias, bem como o produto que se quer desenhar (assinalado na imagem seguinte).

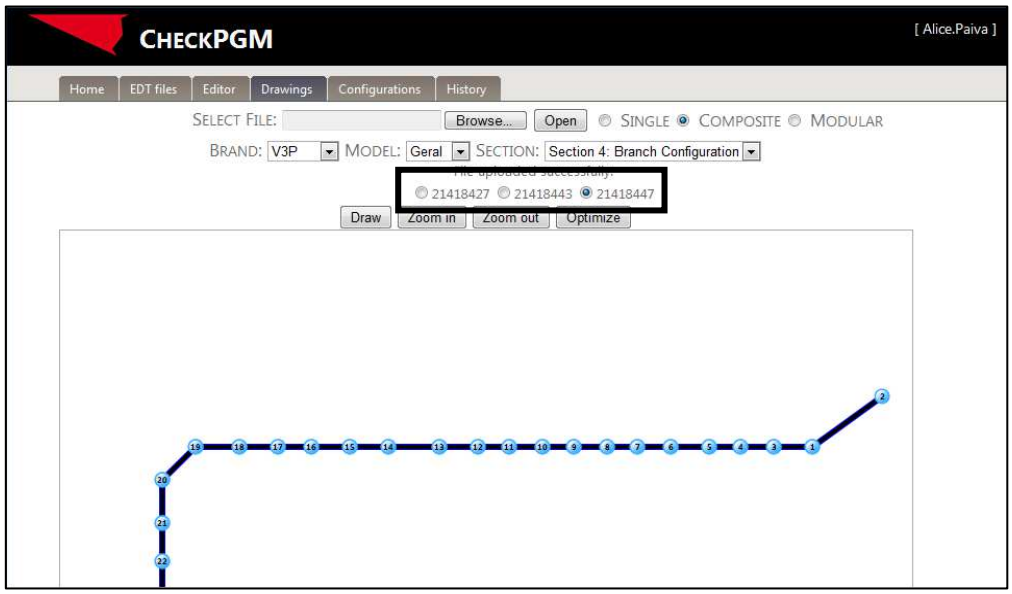


Figura 55: Página de desenho – caso *composite*

- Caso Modular

No caso modular, após o *upload* do ficheiro DSI, é necessário escolher os módulos que constituem a cablagem pretendida (zona da imagem seguinte assinalada a preto). Após ser desenhada, é possível identificar cada módulo na cablagem resultante através do menu assinalado na imagem seguinte a azul. O módulo evidencia-se então a vermelho.

Figura 56: Página de desenho – caso modular

Menu Configurações

O menu das configurações contém ligações a todas as páginas que permitem criar e modificar as configurações da aplicação. Conforme se pode verificar na imagem 57, existem 4 ligações possíveis:

1. Página de configuração de parametrizações;
2. Página de configuração de codificação de cores;
3. Página de gestão de marcas;
4. Página de gestão de permissões;

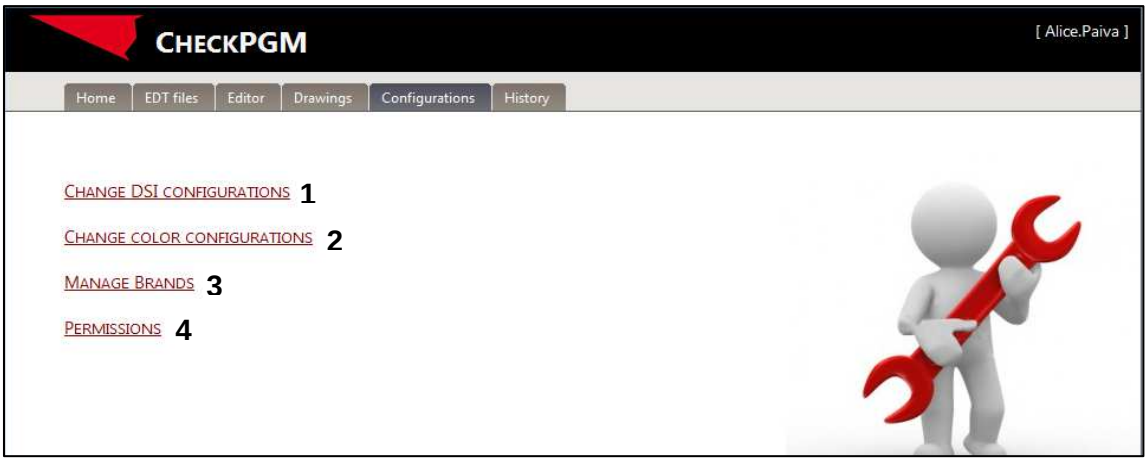


Figura 57: Menu de configurações

- Configurar parametrizações

  Aqui podem-se criar ou alterar as parametrizações de cada marca. A figura seguinte mostra essa mesma página:

Figura 58: Página de configuração de parametrizações

- Configurar codificação de cores

  Aqui podem-se criar ou alterar as codificações de cores de cada marca, de acordo com a imagem seguinte.

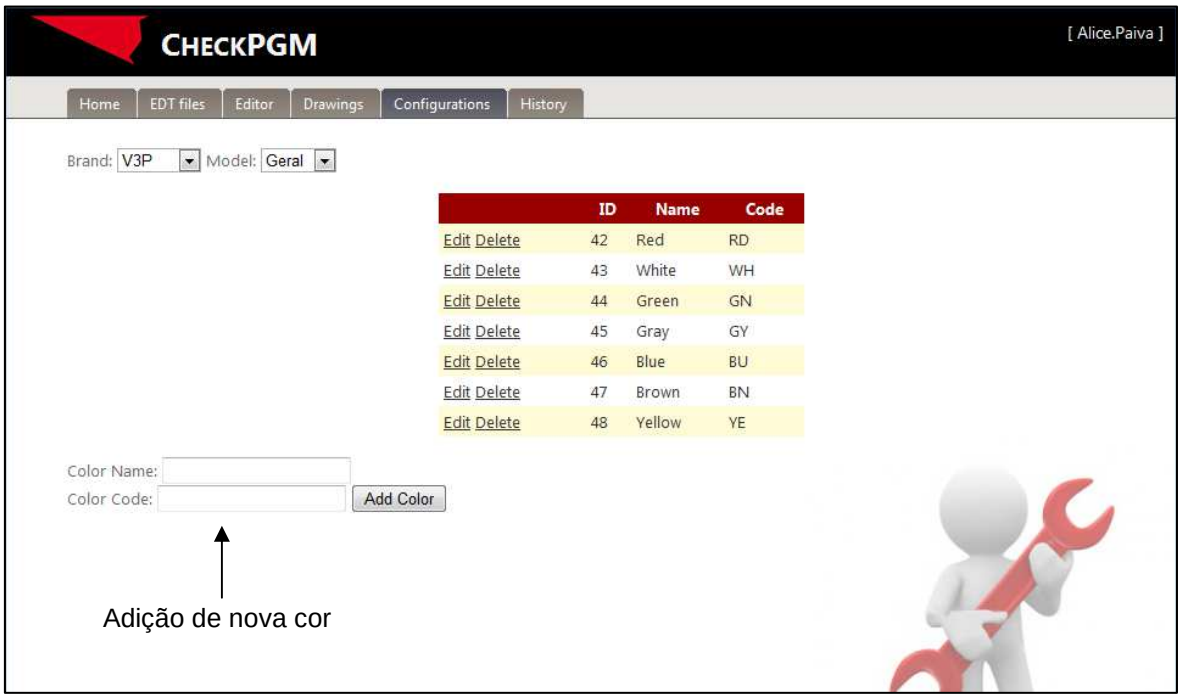


Figura 59: Página de edição da codificação das cores

- Configurar marcas

  Aqui podem-se criar ou alterar marcas e modelos existentes, à semelhança do menu anterior, como se verifica pela figura seguinte:

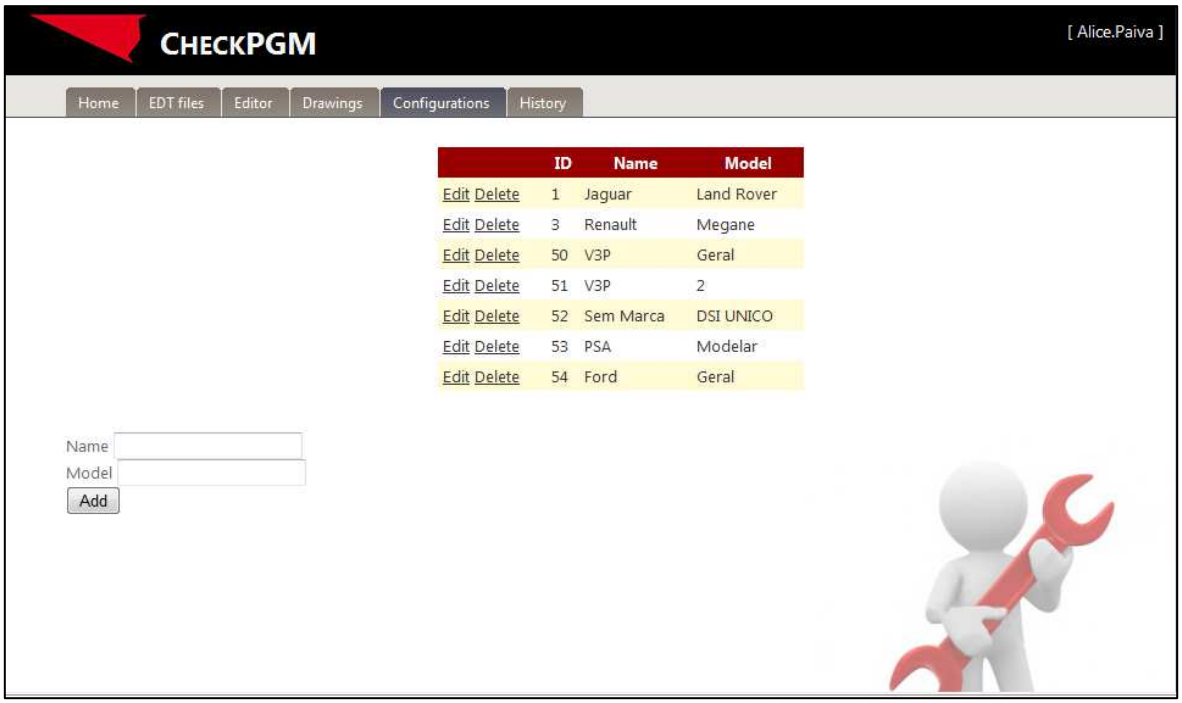


Figura 60: Página de edição de marcas

- Configurar permissões

  Aqui podem-se criar ou alterar as permissões existentes, numa tabela simples, de acordo com a figura 61.

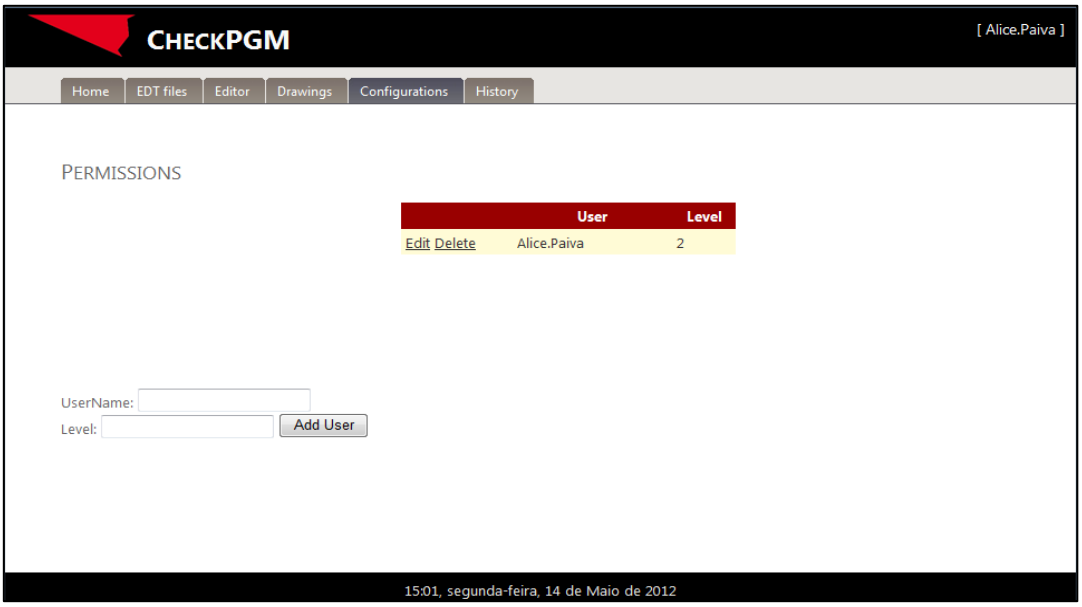


Figura 61: Página de gestão de permissões

Histórico

A página de histórico guarda em base de dados as ações que são feitas na aplicação, com os dados na marca, utilizador, data e o tipo de ação do utilizador atual. Também contém um campo para nota, que pode ser alterado a qualquer momento clicando no texto *edit.* Clicando na hiperligação "*VIEW ALL HISTORY ENTRIES*", apenas acessível a utilizadores com nível 2, poderão ser visualizadas todas as ações de todos os utilizadores, bem como eliminá-las.

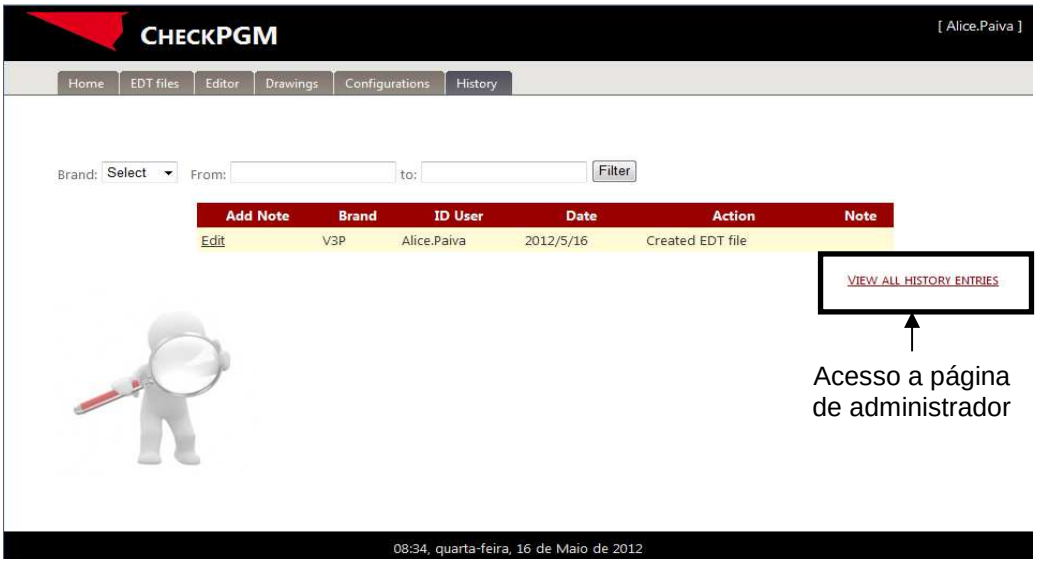


Figura 62: Página de histórico

Capítulo 4 Conclusões

4.1. Kaizen

A Yazaki tem vindo a implementar um conjunto de mudanças na sua estrutura, mudanças essas que se denominam por *New Yazaki System* (NYS). Tem por base o sistema Toyota Production System (TPS) e tem como principal conceito o Kaizen: “do japonês 改 善, **mudança para melhor**, é uma palavra de origem japonesa com o significado de melhoria contínua, gradual, na vida em geral (pessoal, familiar, social e no trabalho)”[11]. Aplicado ao ambiente empresarial traduz-se numa capacidade de reagir a todas as mudanças dos clientes em termos de quantidade e variedade de produtos[5] e em produzir produtos com melhor qualidade, menor custo, de forma mais rápida mas com mais segurança e menos esforço. Nesse sentido fez-se uma comparação entre o cenário existente antes da aplicação ser desenvolvida e depois.

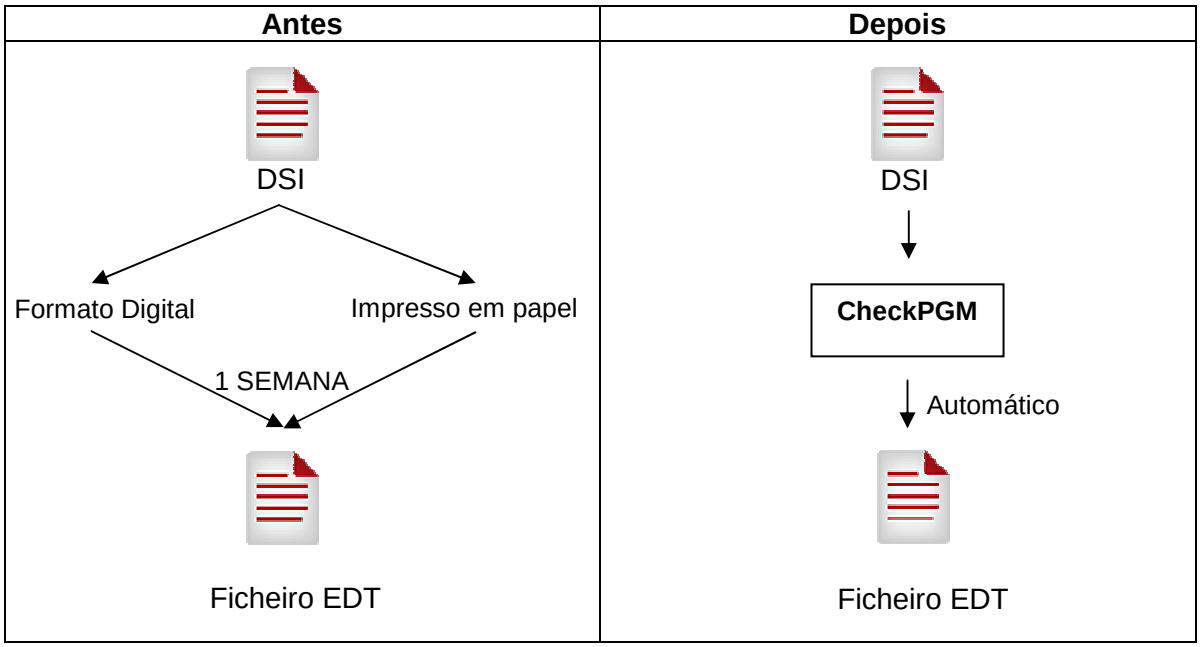


Tabela 11: Kaizen

4.2. Considerações finais

O estágio decorreu com normalidade, sem percalços e o objetivo principal do projeto foi concluído com sucesso.

Na minha opinião o estágio foi uma experiência extremamente enriquecedora a todos os níveis. A nível profissional foi dado um passo importante uma vez que a minha formação académica de base estava um pouco distanciada da área em questão e assim aprendi, com bastante trabalho, novos conceitos e técnicas de programação que até então me eram desconhecidas. Apesar de manipular uma grande variedade de linguagens e métodos simultaneamente ter sido uma tarefa árdua, fiquei familiarizada com um vasto leque de tecnologias, que me é uma mais valia a nível de experiência profissional.

A nível pessoal, o contato com as pessoas e com uma hierarquia empresarial deu-me a conhecer um novo cenário de trabalho, criando uma nova perspetiva sobre os mecanismos que uma empresa necessita para funcionar eficientemente e ter sucesso.

Referências

1. Aho, Alfred; Lam,Monica,Sethi, Ravi;Ullman, Jeffrey; Compilers - Principles, Techniques, & Tools, Second Edition, Person Addison Wesley, 2007.

2. Armstrong, Damon - .NET Application Architecture: the Data Access Layer, Julho 2006 publicado online em:
[http://www.simple-talk.com/dotnet/.net-framework/.net-application-architecture-the-data-access-layer/].

3. Arquitectura em três camadas – Parte 1
[http://marcelamperes.wordpress.com/2011/07/14/arquitetura-em-tres-camadas-parte-1/];

4. Carvalho, Adelaide, Práticas de C#, Programação orientada por objetos, FCA, Julho 2011.

5. Empresa – Manual de formação Yazaki, Outubro de 2011.

6. Ferrari, Denis - Tratamento de erros em asp.net – Fevereiro de 2009
[http://aprenderaspx.wordpress.com/2009/02/22/tratamento-de-erros-em-projetos-aspnet/].

7. Fluxo de componentes – Manual de formação Yazaki, Outubro de 2011.

8. Friedl, Jeffrey – Mastering Regular Expressions,3rd Edition, O'Reilley, 2006.

9. Intranet e extranet, 28/07/2009
[http://pt.kioskea.net/contents/entreprise/intranet.php3].

10. JavaScript
[http://pt.wikipedia.org/wiki/JavaScript].

11. Kaizen
[http://pt.wikipedia.org/wiki/Kaizen].

12. Kambalyal, Channu – *3-tier Architecture, 2010*, publicado online em:
[http://www.research55.com/video_docu/21qanda77NTierArchitecture.pdf].

13. Leite, Mário - Programação Orientada ao Objeto: uma abordagem didática, artigo publicado online em:
[http://www.ccuec.unicamp.br/revista/infotec/artigos/leite_rahal.html].

14. Modelo em três camadas – Wikipedia
[http://pt.wikipedia.org/wiki/Modelo_em_tr%C3%AAs_camadas].

15. .NET Framework
[http://en.wikipedia.org/wiki/.NET_Framework].

16. Gorton, Ian – *Essential Software Architecture*, Springer, 2006.

17. Pereira, Alexandre; Poupa, Carlos - Linguagens Web, 4ª edição, 2011, Edições Sílabo.

18. Pinto Brasil, Fábrica de Máquinas Industriais
[http://www.pintobrasil.pt].

19. Three Layer Architecture in C# .NET publicado online em:
[http://www.codeproject.com/Articles/36847/Three-Layer-Architecture-in-C-NET].

20. WireIt documentation
[http://neyric.github.com/wireit/api/index.html].

21. Yazaki-Europe – Site oficial
[http://www.yazaki-europe.com].

22. Cardoso, Domingos, Szymanski, Rostami, Mohammad - Matemática Discreta, Escolar Editora, 2009.

ANEXO A: Glossário

- Bucha - Acessório para vedar da água, pó e humidade;

- *Cap* - Acessório que proteje o *joint*;

- Cablagem - Conjunto de circuitos (fios condutores) e componentes; que tem como principal papel coordenar e controlar toda a distribuição do sistema elétrico do automóvel;

- *Checker* – painel de inspeção elétrica;

- *Clip* - Componente que fixa a cablagem à carroçaria do automóvel;

- Conetor - Componente que isola a corrente elétrica com o exterior e entre os vários terminais nele introduzidos. Também serve de ligação aos instrumentos elétricos do automóvel e de cablagem para cablagem;

- Conetor de *joint* - Faz a junção entre vários circuitos;

- Coto - Acessório para proteger de agressões mecânicas e físicas;

- Desfolha - Permite a passagem de corrente elétrica do fio para o terminal, quando este é cravado;

- Desfolha intermédia - Permite fazer a ligação de um ou mais circuitos através da cravação de *joint* de terminal;

- Equipotencial/*Equipotential* - Valor de potencial de um fio;

- Fio - Condutor de corrente. Existem várias espécies, secções e cores (simples ou composta). A cor mais predominante aparece em primeiro lugar;

- Fusível - Peça que protege os circuitos e os órgãos elétricos do automóvel, em caso de sobrecarga do sistema;

- *Joint* - Junção;

- Malha - Tem como função eliminar as interferências radioelétricas;

- *Shrink* - Isolar as junções e a cravação de cabos de bateria de secção elevada;

- Terminal - Permite ligar entre si os diferentes circuitos;

- Vinil - Acessório que protege e/ou isola;

ANEXO B: Bibliotecas externas

Conjunto de bibliotecas de desenhos de grafos resultantes da pesquisa online:

- GraphViz - Graph Visualization Software
  http://www.graphviz.org/;

GraphViz é um software open-source de visualização de grafos escrita na linguagem DOT (Linguagem de descrição de grafos). O software recebe descrições de grafos em texto simples e cria grafos em formatos úteis, tal como PDF, Postscript, ou para visualização num browser. Tem opções para cores, letras, estilos, hiperligações, entre outros.

- QuickGraph - Generic Graph Data Structures and Algorithms for .NET
  http://www.codeproject.com/Articles/5603/QuickGraph-A-100-C-graph-library-with-Graphviz-Sup

Quickgraph contém estruturas de dados para grafos direcionados e não direcionados para a plataforma .NET. A biblioteca traz incorporados algoritmos tais como pesquisa em profundidade, caminho mais curto, fluxo máximo, árvore abrangente. É uma biblioteca em C# que contém suporte para várias outras bibliotecas, como por exemplo o GraphViz.

- GrapheNET (http://graphenet.codeplex.com/documentation)

GrapheNET é outra biblioteca para a plataforma .NET para a representação de grafos em C# . Não existe muita documentação sobre esta biblioteca, pelo que a sua utilização não tem um suporte online útil.

- MSAGL (GLEE) - Microsoft Automatic Graph Layout (http://chalaki.com/how-to-program-msagl-glee-to-create-hierarchical-graph-layouts/519)

É uma biblioteca para .NET desenvolvida pela Microsoft para a criação de grafos em bitmaps. É constituído por três componentes integradas:

- Motor de esquemas automático que permite a criação de árvores;
- Camada de desenho para permitir a modificação dos atributos de cada componente, como a forma dos nós ou o estilo das arestas;
- Um visualizador baseado no Windows para criar grafos interativos.

Anexo C: Lista telefónica para VoIP

Paralelamente ao projeto principal, foi desenvolvida uma aplicação mais simples chamada VoIP, que tem como objetivo principal gerir a lista telefónica entre empresas Yazaki.

- **Contextualização**

A Yazaki é constituída é uma empresa difundida por vários países do mundo, pelo que a comunicação entre as diferentes localizações é algo bastante frequente. Para isso, utilizam um sistema de VoIP (Voz por IP) que utiliza a internet para fazer chamadas telefónicas. Criou-se no passado uma folha de cálculo de consulta que contém todos esses contatos que são em elevado número.

- **Objetivo**

Para consultar um dado contato telefónico era necessário aceder a uma folha de cálculo e fazer uma pesquisa manual. O objetivo desta aplicação é simplificar esta pesquisa e fornecer as devidas ferramentas de manutenção dos contatos. Dado que o sistema funciona através de endereços de IP, tornou-se interessante reconhecer a localização da origem da chamada selecionando automaticamente a mesma nas opções de pesquisa.

- **Modelo de Dados**

Criou-se então uma base de dados para o efeito seguindo o modelo relacional da figura seguinte:

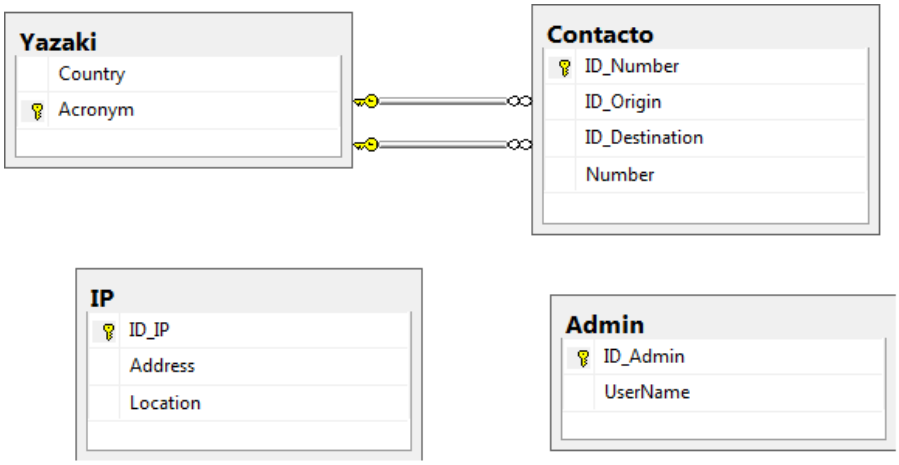


Figura 1: Modelo de domínio da aplicação VoIP

Como se pode ver na figura 1, o modelo é bastante simples contento duas tabelas principais: Yazaki, que contém o país e a sigla de cada localização Yazaki (por exemplo, em Portugal uma das siglas possíveis é YSE); e a tabela Contacto que contém a origem, destino e número telefónico correspondente, fora o identificador de cada registo. A tabela IP , como o nome indica contém todos os endereços de IP existentes na empresa, tendo como localização a sigla da empresa correspondente. A tabela de administradores, Admin, apenas contém um identificador de registo e o nome de utilizador Yazaki.

- **Permissões**

A aplicação reconhece dois tipos de utilizadores: os administradores e os restantes. Os administradores têm acesso ao separador do menu que contém as opções de edição, enquanto que os restantes não têm essa opção visível.

- **Interface**

O interface da aplicação é bastante intuitivo e simples. Funciona através de tabelas que mostram a informação da base de dados. Como é feito um reconhecimento de endereço IP, a localização de origem é selecionada automaticamente, como está assinalado na figura.

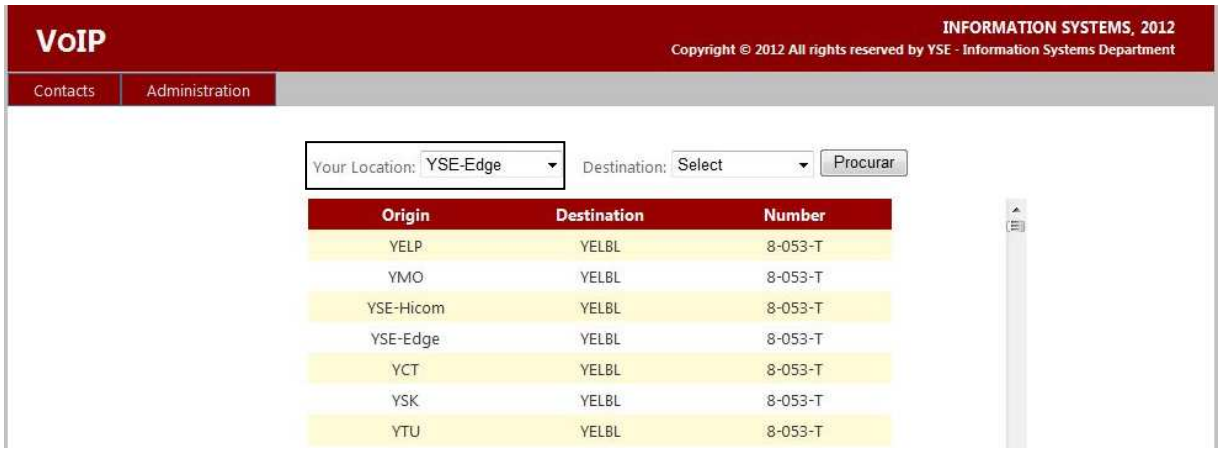


Figura 2: Listagem de contactos inicial

Adicionalmente a esta listagem, existe um separador de administrador que fornece todas as ferramentas de edição de contactos, como se pode ver na figura 3.

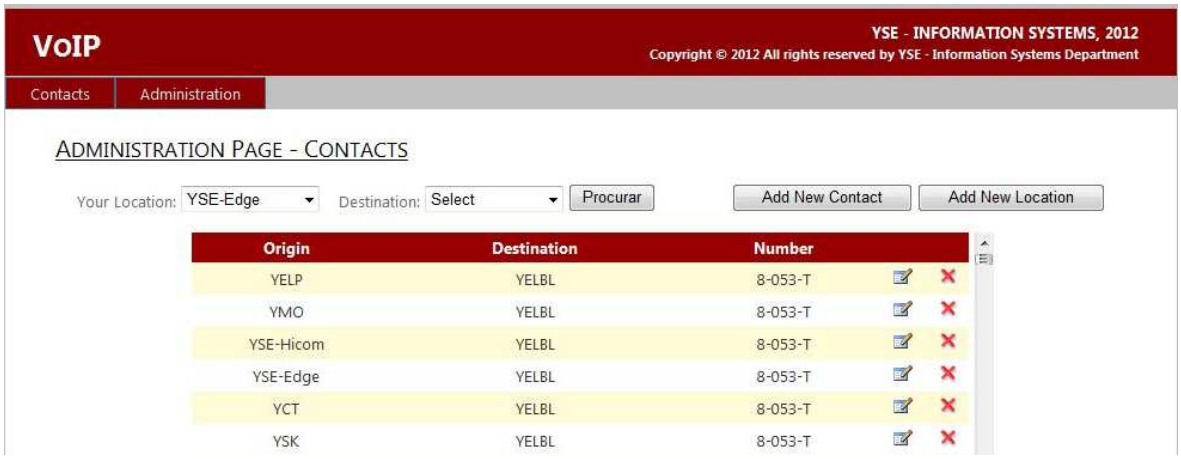


Figura 3: Listagem de contactos na página de administração

Na figura 4 mostra-se a página onde é possível adicionar novos administradores e alterar os existentes. O nome de utilizador usado é o mesmo dado pelo nome de utilizador

interno da empresa e é reconhecido automaticamente através de funções existentes em componentes já programadas pelo setor de Sistemas de Informação.

Figura 4: Página de edição de administradores

Na figura seguinte é mostrada a página de manutenção de endereços de IP, que funciona à semelhança das páginas mostradas anteriormente.

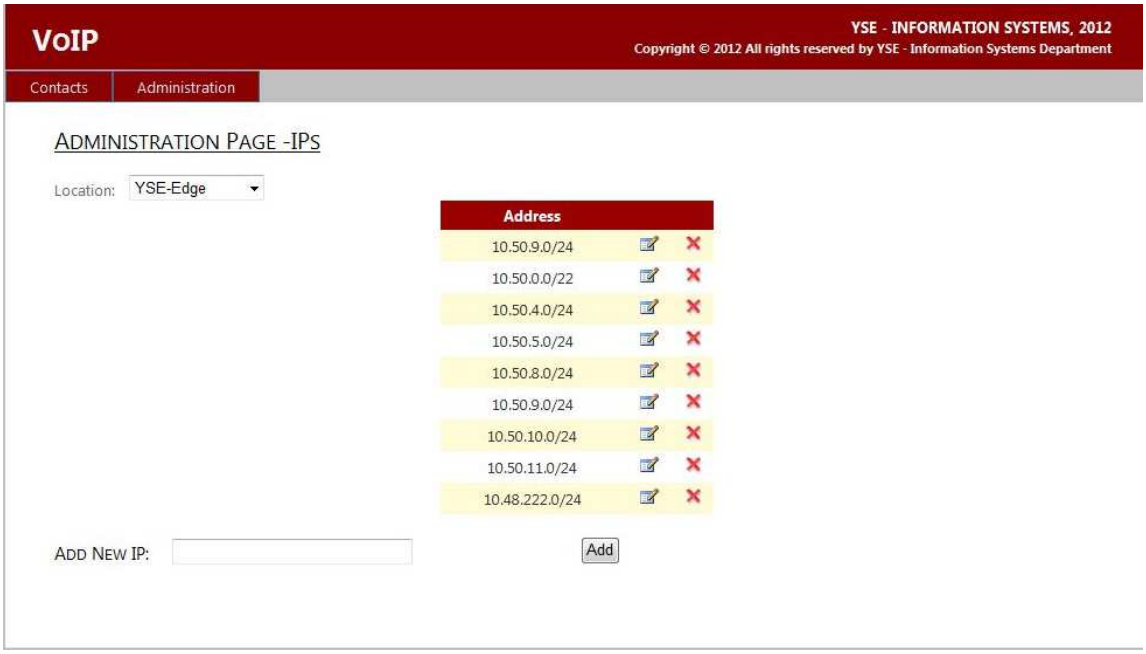


Figura 5: Página de administração de endereços IP